

Problem Theory

Ramón Casares

ORCID: [0000-0003-4973-3128](https://orcid.org/0000-0003-4973-3128)

*The Turing machine, as it was presented by Turing himself, models the calculations done by a person. This means that we can compute whatever any Turing machine can compute, and therefore we are Turing complete. The question addressed here is why, Why are we Turing complete? Being Turing complete also means that somehow our brain implements the function that a universal Turing machine implements. The point is that evolution achieved Turing completeness, and then the explanation should be evolutionary, but our explanation is mathematical. The trick is to introduce a mathematical theory of problems, under the basic assumption that solving more problems provides more survival opportunities. So we build a problem theory by fusing set and computing theories. Then we construct a series of resolvers, where each resolver is defined by its computing capacity, that exhibits the following property: all problems solved by a resolver are also solved by the next resolver in the series if certain condition is satisfied. The last of the conditions is to be Turing complete. This series defines a resolvers hierarchy that could be seen as a framework for the evolution of cognition. Then the answer to our question would be: *to solve most problems*. By the way, the problem theory defines adaptation, perception, and learning, and it shows that there are just three ways to resolve any problem: routine, trial, and analogy. And, most importantly, this theory demonstrates how problems can be used to found mathematics and computing on biology.*

Keywords: *problem solving; adaptation, perception & learning; Turing completeness; resolvers hierarchy; evolution of cognition.*

Problem Theory

| | |
|------------------------------|-----------|
| §1 Introduction | 3 |
| §1.1 Object | 3 |
| §1.2 Contents | 4 |
| §2 Theory | 5 |
| §2.1 Problem | 5 |
| §2.2 Solution | 6 |
| §2.3 Resolution | 6 |
| §2.4 Eight Concepts | 7 |
| §3 Sets | 7 |
| §3.1 Problems | 7 |
| §3.2 Solutions | 9 |
| §3.3 Routines and Trials | 11 |
| §3.4 Analogies | 12 |
| §3.5 Metaproblems | 14 |
| §3.6 Resolution Typology | 16 |
| §4 Computers | 17 |
| §4.1 Turing Machine | 17 |
| §4.2 Turing Completeness | 19 |
| §4.3 Turing's Thesis | 22 |
| §4.4 Full Resolution Machine | 24 |
| §4.5 Problem Topology | 27 |
| §5 Resolvers | 29 |
| §5.1 Semantics and Syntax | 29 |
| §5.2 Mechanism | 31 |
| §5.3 Adapter | 32 |
| §5.4 Perceiver | 34 |
| §5.5 Learner | 36 |
| §5.6 Subject | 38 |
| §5.7 Resolvers Hierarchy | 39 |
| §6 Conclusion | 41 |
| §6.1 Purpose | 41 |
| §6.2 Countability | 42 |
| §6.3 Intuition | 43 |
| References | 44 |

§1 Introduction

Devoid of problems, thinking is useless.

WARNING This paper does not explain how to solve, nor how to resolve, any problem.

§1.1 Object

¶1 · The object of this paper is to present a mathematical theory of problems. The resulting problem theory provides meaning to set theory and to computing theory.

¶2 · Problems are nearly everywhere. We can say that mathematics is all about mathematical problems, but also that physics is all about physical problems, and philosophy is all about philosophical problems. I said nearly because there are not problems in a river; a river just flows. So, where are problems?

¶3 · This problem theory gives an answer: There are problems where there is freedom. Determinists will surely object, but they should note that if there were only uncertainty, and not the possibility of doing otherwise, then problem resolving would be purposeless and absurd. Nevertheless, in this theory freedom cannot exist by itself, but freedom is always limited by a condition and both together, freedom and a condition, are a problem. In fact, the resolution of any problem is the process of spending all of its freedom while still satisfying the condition. So resolving is fighting freedom away. And, if people fight for freedom, it is because we want problems; in fact, not having any problem is boring. But I would say more, we are devices exquisitely selected to resolve problems, because surviving is literally the problem of being dead or alive: “To be, or not to be—that is the question.”

¶4 · I am digressing, sorry! The point is that problems are related to sets at the very bottom: for each problem there is a condition that determines if anything is a solution to it or not, so for each problem there is a set, the set of its solutions, and the condition is its characteristic function. This means that problems and sets are just two names for the same thing. So problem theory, being just a rewording of set theory, would be a better foundation for mathematics than set theory, because problems are more related to thinking than sets are.

¶5 · We have just seen how problems and solutions fit with sets, but we have seen nothing about resolutions, that is, the ways to go from a problem to its solutions. It is a fact that computing is helping us in resolving many problems. Perhaps too many: How our modern society would subsist without computers? I am digressing again, sorry! The right question is: What is the relation between problem resolving and computing?

¶6 · Computing is the mechanical manipulation of strings of symbols. Mechanical in the sense that the manipulations do not take into account the meaning of the symbols, but they just obey blindly a finite set of well-defined rules. Being meaningless, what could be the purpose of computing? Historically, computing resulted from two apparently different pursuits: the foundation of mathematics, and the enhancement of calculating machines. The second, the development of mechanical aids to calculation, is easier to understand. When we learn the algorithm for division we readily appreciate that those fixed rules can be better applied by a machine than by a person. This explains why an arithmetic calculator comes handy when resolving a problem that requires performing a numerical division. And it could also help us to understand why computation was seen as the ideal for mathematical rigor, and then how computing relates to the foundations of mathematics.

¶7 · But, again, what is the purpose of mathematical formalization? Is it true that a complete formalization of mathematics would render it meaningless? What would be the use of something meaningless? And again, the arithmetic calculator, dividing for us, answers the three questions: formalization prevents mistakes and assures that nothing has been taken for granted, and, while it is literally meaningless, it is not useless if the formalism helps us in resolving problems. Though pending on an if, formalism is not yet lost. This paper cuts that Gordian knot by showing that problem resolving is computing.

¶8 · In fact, that ‘resolving is computing’ comes from the founding paper of computing. Turing (1936) proved that the *Entscheidungsproblem*, which is the German word for ‘decision problem’, is unsolvable, because it cannot be solved by any Turing machine. For this proof to be valid, ‘solved by a Turing machine’ has to be equal to ‘solved’, and therefore ‘resolved by computing’ has to be redundant.

¶9 · Summarizing, a problem is a set, and resolving is computing. This is how this problem theory relates to set and computing theories at the highest level of abstraction. For a more detailed view you should continue reading this paper.

§1.2 Contents

¶1 · The object of this paper is to introduce a mathematical theory of problems. Because our approach is minimalist, aiming to keep only what is essential, we will define a problem theory from first principles. Section §2 contains this problem theory, including its eight concepts: problem, with freedom and condition; resolution, with routine, trial and analogy; and solution. Some care is advisable to distinguish ‘solution’ from ‘resolution’, because while they are usually considered synonyms, they are very distinct concepts in this theory: a solution is a state, and a resolution is a transition. Then that ‘a problem is resolved unsolvable’ achieves a very precise meaning.

¶2 · Section §3 translates the problem theory to set theory. Subsection §3.1 defines what a problem is, and what is the set of its solutions is defined in Subsection §3.2. Then, in Subsection §3.3, we develop the first two ways to resolve a problem, by routine and by trial, while we devote Subsection §3.4 to the third way, by analogy. The conclusion of these two subsections is that there is a general form that includes the three forms. Then we observe that looking for a resolution to a problem is also a problem, the metaproblem, so Subsection §3.5 deals with metaproblems. The last subsection of this section, Subsection §3.6, shows that there is only one level of problem meta-ness and that there are five types of resolution.

¶3 · The next section, Section §4, is about computing. In Subsection §4.1 we present the Turing machine, concluding that all computing is inside countable sets. In Subsection §4.2 we deal with universal computers and Turing completeness. Then, in Subsection §4.3, we explain that Turing’s thesis implies that everything is an expression, that resolving is computing, and that all problem sets are countable. In Subsection §4.4, we introduce the full resolution machine, and we show some equivalences between problem theory and computing theory. In the last subsection of this section, Subsection §4.5, we show that there are five types of problem.

¶4 · Section §5 is about resolvers, that is, devices that resolve problems. In the first subsection, §5.1, we present the practical scenario, where functions are not solutions, so we distinguish the semantics of solutions from the syntax of functions, and we define the range of a resolver as the set of problems that the resolver solves, and the power of a

resolver as the set of problems that the resolver resolves. Then we construct a series of five resolvers:

- Mechanism, Subsection §5.2, is any device that implements a semantic unconditional computation. We show that mechanisms can resolve problems by routine.
- Adapter, in §5.3, is any device that implements a semantic conditional computation. We show that adapters can resolve problems by trial.
- Perceiver, in §5.4, is any device that implements a semantic functional computation or a syntactic unconditional computation. We show that perceivers can resolve problems by analogy and metaproblems by routine.
- Learner, in §5.5, is any device that implements a syntactic conditional computation. We show that learners can resolve metaproblems by trial.
- Subject, in §5.6, is any device that implements a syntactic functional computation. We show that subjects can resolve metaproblems by analogy.

In addition, we show that the range and power of each resolver in the series includes the range and power of the previous one, provided that a specific condition is satisfied. The last of these conditions requires the subject to be Turing complete. So, in the last subsection of this section, §5.7, we summarize the findings of the section: we show that there is a hierarchy of five types of resolver, and that the problem theory is complete. The theory is complete because Turing completeness is the maximum computing capacity, and this means that there are exactly three ways to resolve any problem: routine, trial, and analogy. Finally, we argue that we are the Turing complete subjects that have resulted from an evolution of resolvers of the survival problem.

¶5 · The paper finishes with some conclusions, in Section §6. In the first subsection, §6.1, we explain how problem theory provides purpose and meaning to set theory and to computing theory. In the next subsection, §6.2, we argue that countableness is the golden mean that keeps paradoxes under control. And in the last subsection, §6.3, we explore what would be the implications of non-computable ways of resolving, as intuition.

§2 Theory

§2.1 Problem

¶1 · Every *problem* is made up of freedom and of a condition. There have to be possibilities and *freedom* to choose among them, because if there is only necessity and fatality, then there is neither a problem nor is there a decision to make. The different possible options could work, or not, as solutions to the problem, so that in every problem a certain *condition* that will determine if an option is valid or not as a solution to the problem must exist.

$$\text{Problem} \begin{cases} \text{Freedom} \\ \text{Condition} \end{cases}$$

§2.2 Solution

¶1 · A fundamental distinction that we must make is between the solution and the resolution of a problem. Resolving is to searching as solving is to finding, and please note that one can search for something that does not exist.

Resolving · Searching
Solving · Finding

Thus, *resolution* is the process that attempts to reach the solutions to the problem, while a *solution* of the problem is any use of freedom that satisfies the condition. In the state-transition jargon: a problem is a state of ignorance, a solution is a state of satisfaction, and a resolution is a transition from uncertainty to certainty.

Problem $\xrightarrow{\text{Resolution}}$ Solution

¶2 · We can explain this with another analogy. The problem is defined by the tension that exists between two opposites: freedom, free from any limits, and the condition, which is pure limit. This tension is the cause of the resolution process. But once the condition is fulfilled and freedom is exhausted, the solution annihilates the problem. The resolution is, then, a process of annihilation that eliminates freedom as well as the condition of the problem, in order to produce the solution.

$$\underbrace{\left. \begin{array}{l} \text{Freedom} \\ \text{Condition} \end{array} \right\}}_{\text{Problem}} \xrightarrow{\text{Resolution}} \text{Solution}$$

¶3 · A mathematical example may also be useful in order to distinguish resolution from solution. In a problem of arithmetical calculation, the solution is a number and the resolution is an algorithm such as the algorithm for division, for example.

§2.3 Resolution

¶1 · There are three ways to resolve a problem: routine, trial, and analogy.

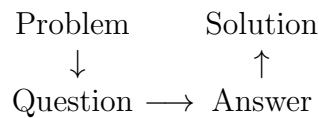
Resolution $\left\{ \begin{array}{l} \text{Routine} \\ \text{Trial} \\ \text{Analogy} \end{array} \right.$

¶2 · To resolve a problem by *routine*, that is, by knowing or without reasoning, it is necessary to know the solutions, and it is necessary to know that they solve that problem.

¶3 · If the solutions to a problem are not known, but it is known a set of possible solutions, then we can use a trial and error procedure, that is, we can try the possible solutions. To resolve by *trial* is to test each possible solution until the set of possible solutions is exhausted or a halting condition is met. There are two tasks when we try: to test if a particular possibility satisfies the problem condition, and to govern the process determining the order of the tests and when to halt. There are several ways to govern the process, that is, there is some freedom in governing the trial, and so, if we also put

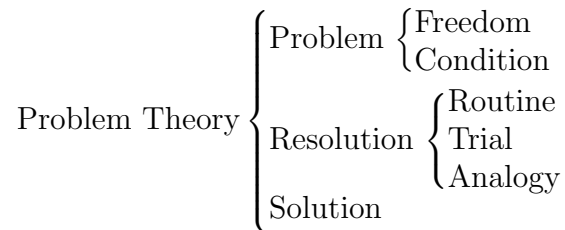
a condition on it, for example a temporal milestone, then governing is a problem. And there are three ways to resolve a problem (*da capo*).

¶4 · By *analogy* we mean to transform a problem into a different one, called question, which is usually composed of several subproblems. This works well if the subproblems are easier to resolve than the original problem. There are usually several ways to transform any problem (there is freedom), but only those transformations that result in questions that can be resolved are valid (which is a condition), so applying an analogy to a problem is a problem. There are three ways to resolve the analogy, the question, and each of its subproblems: routine, trial, and analogy (*da capo*). If we could translate a problem into an analogue question, and we could find a solution to that question, called answer, and we could perform the inverse translation on it, then we would have found a solution to the original problem.



§2.4 Eight Concepts

¶1 · Lastly we are ready to list the eight concepts of the problem theory. They are: problem, with freedom and condition; resolution, with routine, trial, and analogy; and solution.



§3 Sets

§3.1 Problems

§3.1.1 NOTATION We will refer to the set of problems as \mathbb{P} . We will refer to the set of resolutions as \mathbb{R} . We will refer to the set of solutions as \mathbb{S} .

DEFINITION A resolution takes a problem and returns the set of the solutions to the problem. Then resolutions are $\mathbb{R} = \mathbb{P} \rightarrow 2^{\mathbb{S}}$, where $2^{\mathbb{S}}$ is the powerset, or the set of the subsets, of \mathbb{S} .

§3.1.2 NOTATION \top stands for ‘true’, and \perp for ‘false’. We will refer to the set of these Boolean values as \mathbb{B} . $\mathbb{B} = \{\top, \perp\}$.

COMMENT $\top = \neg\perp$ and $\perp = \neg\top$. Also $[P = \top] = P$ and $[P = \perp] = \neg P$.

§3.1.3 NOTATION Given $s \in S \subseteq \mathbb{S}$ and $f \in F \subseteq (\mathbb{S} \rightarrow \mathbb{S})$, so $f : \mathbb{S} \rightarrow \mathbb{S}$ and $f(s) \in \mathbb{S}$, we will use the following rewriting rules:

$$f(S) = \{f(s) \mid s \in S\}, F(s) = \{f(s) \mid f \in F\}, \text{ and } F(S) = \{f(s) \mid s \in S \times f \in F\}.$$

COMMENT As $f(s) \in \mathbb{S}$, then $f(S) \in 2^{\mathbb{S}}$, $F(s) \in 2^{\mathbb{S}}$, and $F(S) \in 2^{\mathbb{S}}$.

PROPOSITION If $s \in S$ and $f \in F$, then $f(S) \subseteq F(S)$ and $F(s) \subseteq F(S)$.

§3.1.4 DEFINITION Problem π is $x?P_\pi(x)$, where P_π is any predicate, or Boolean-valued function, on \mathbb{S} ; so $P_\pi : \mathbb{S} \rightarrow \mathbb{B}$, where $P_\pi(x) = \top$ means that x is a solution of π , and $P_\pi(x) = \perp$ means that x is not a solution of π .

COMMENT A problem $\pi = x?P_\pi(x)$ is made up of freedom and of a condition, as defined in Section §2. The condition is P_π , and freedom is represented by the free variable x , which is free to take any value in \mathbb{S} , $x \in \mathbb{S}$.

§3.1.5 DEFINITION A function $*f$ is effectively calculable if there is a purely mechanical process to find $*f(s)$ for any s .

COMMENT This definition of effective calculability was stated by Turing (1938), §2.

COMMENT If the result of the calculation is finite, then an effective calculation has to complete it. If the result of the calculation is infinite, then an effective calculation has to proceed forever towards the result.

NOTATION We will refer to the set of effectively calculable functions as $*\mathbb{F}$.

§3.1.6 DEFINITION A problem π is expressible if its condition P_π is an effectively calculable function.

COMMENT The result of a condition is in set $\mathbb{B} = \{\top, \perp\}$, so it is always finite. Therefore a problem is not expressible if for some x we cannot calculate whether x is a solution or not in a finite time.

§3.1.7 DEFINITION The condition isomorphism is the natural isomorphism that relates each problem π with its condition P_π : for each predicate P there is a problem, $x?P(x)$, and for each problem, $\pi = x?P_\pi(x)$ there is a predicate, P_π . That is, $\mathbb{P} \Leftrightarrow (\mathbb{S} \rightarrow \mathbb{B}) : x?P_\pi(x) \Leftrightarrow P_\pi$.

COMMENT Using the condition isomorphism, two problems are equal if they have the same condition, that is, $\pi = \rho \Leftrightarrow P_\pi = P_\rho$.

COMMENT The condition isomorphism abstracts freedom away.

§3.1.8 THEOREM The set of problems is the set of predicates, that is, $\mathbb{P} = \mathbb{S} \rightarrow \mathbb{B}$.

PROOF $\mathbb{P} \cong \mathbb{S} \rightarrow \mathbb{B}$, by the condition isomorphism, see §3.1.7, and, abstracting freedom, $\mathbb{P} = \mathbb{S} \rightarrow \mathbb{B}$. But freedom has to be abstracted away from mathematics because freedom is free of form and it cannot be counted nor measured. \diamond

COMMENT Although in mathematics we cannot deal with freedom, it is an essential part of problems, see §2.1. In any case, what defines problem π is its condition P_π .

§3.1.9 LEMMA The name of the free variable is not important, it can be replaced: $x?P(x) = y?P(y)$.

PROOF By the condition isomorphism, and §3.1.8, both problems, $x?P(x)$ and $y?P(y)$, are equal, $x?P(x) = y?P(y)$, because they have the same condition, P . \diamond

COMMENT This means that the rule of α -conversion stands for problem expressions. See Curry & Feys (1958), Section 3D.

§3.1.10 DEFINITION Let π and ρ be two problems. Then $\pi \wedge \rho = x?P_\pi(x) \wedge P_\rho(x)$, and $\pi \vee \rho = x?P_\pi(x) \vee P_\rho(x)$, and $\bar{\pi} = x?\neg P_\pi(x)$.

COMMENT In other words, $P_{\pi \wedge \rho}(x) = P_\pi(x) \wedge P_\rho(x)$, $P_{\pi \vee \rho}(x) = P_\pi(x) \vee P_\rho(x)$, and $P_{\bar{\pi}}(x) = \neg P_\pi(x)$.

COMMENT This provides a way to compose, or decompose, problems.

§3.1.11 DEFINITION A problem τ is tautological if its condition is a tautology; P_τ is a tautology, if $\forall x, P_\tau(x) = \top$. A problem $\bar{\tau}$ is contradictory if its condition is a contradiction; $P_{\bar{\tau}}$ is a contradiction, if $\forall x, P_{\bar{\tau}}(x) = \perp$.

LEMMA Both τ and $\bar{\tau}$ are expressible.

PROOF Because P_τ and $P_{\bar{\tau}}$ are effectively calculable, see §3.1.6 and §3.1.5. \diamond

§3.1.12 THEOREM $\langle \mathbb{P}, \vee, \wedge, \neg, \bar{\tau}, \tau \rangle$ is a Boolean algebra, where $\bar{\tau}$ is the neutral for \vee , and τ is the neutral for \wedge .

PROOF Because $P_\pi(x) \in \mathbb{B}$. In detail, $\forall \pi, \rho, \sigma \in \mathbb{P}$:

- 1o. $(\pi \vee \rho) \vee \sigma = x? P_{\pi \vee \rho}(x) \vee P_\sigma(x) = x? (P_\pi(x) \vee P_\rho(x)) \vee P_\sigma(x) = x? P_\pi(x) \vee (P_\rho(x) \vee P_\sigma(x)) = x? P_\pi(x) \vee P_{\rho \vee \sigma}(x) = \pi \vee (\rho \vee \sigma)$.
- 1a. $(\pi \wedge \rho) \wedge \sigma = x? P_{\pi \wedge \rho}(x) \wedge P_\sigma(x) = x? (P_\pi(x) \wedge P_\rho(x)) \wedge P_\sigma(x) = x? P_\pi(x) \wedge (P_\rho(x) \wedge P_\sigma(x)) = x? P_\pi(x) \wedge P_{\rho \wedge \sigma}(x) = \pi \wedge (\rho \wedge \sigma)$.
- 2o. $\pi \vee \rho = x? P_\pi(x) \vee P_\rho(x) = x? P_\rho(x) \vee P_\pi(x) = \rho \vee \pi$.
- 2a. $\pi \wedge \rho = x? P_\pi(x) \wedge P_\rho(x) = x? P_\rho(x) \wedge P_\pi(x) = \rho \wedge \pi$.
- 3o. $\pi \vee \bar{\tau} = x? P_\pi(x) \vee P_{\bar{\tau}}(x) = x? P_\pi(x) \vee \perp = x? P_\pi(x) = \pi$.
- 3a. $\pi \wedge \tau = x? P_\pi(x) \wedge P_\tau(x) = x? P_\pi(x) \wedge \top = x? P_\pi(x) = \pi$.
- 4o. $\pi \vee \bar{\pi} = x? P_\pi(x) \vee P_{\bar{\pi}}(x) = x? P_\pi(x) \vee \neg P_\pi(x) = x? \top = x? P_\tau(x) = \tau$.
- 4a. $\pi \wedge \bar{\pi} = x? P_\pi(x) \wedge P_{\bar{\pi}}(x) = x? P_\pi(x) \wedge \neg P_\pi(x) = x? \perp = x? P_{\bar{\tau}}(x) = \bar{\tau}$.
- 5o. $\pi \vee (\rho \wedge \sigma) = x? P_\pi(x) \vee P_{\rho \wedge \sigma}(x) = x? P_\pi(x) \vee (P_\rho(x) \wedge P_\sigma(x)) = x? (P_\pi(x) \vee P_\rho(x)) \wedge (P_\pi(x) \vee P_\sigma(x)) = x? P_{\pi \vee \rho}(x) \wedge P_{\pi \vee \sigma}(x) = (\pi \vee \rho) \wedge (\pi \vee \sigma)$.
- 5a. $\pi \wedge (\rho \vee \sigma) = x? P_\pi(x) \wedge P_{\rho \vee \sigma}(x) = x? P_\pi(x) \wedge (P_\rho(x) \vee P_\sigma(x)) = x? (P_\pi(x) \wedge P_\rho(x)) \vee (P_\pi(x) \wedge P_\sigma(x)) = x? P_{\pi \wedge \rho}(x) \vee P_{\pi \wedge \sigma}(x) = (\pi \wedge \rho) \vee (\pi \wedge \sigma)$.

\diamond

§3.2 Solutions

§3.2.1 THEOREM Everything is in \mathbb{S} . In other words, \mathbb{S} is the set of everything.

PROOF Anything, let us call it s , is a solution to problem $x? [x = s]$, because equality is reflexive, and therefore everything satisfies the condition of being equal to itself. \diamond

COMMENT Freedom is complete, because x is free to take any value; $x \in \mathbb{S}$ is not a restriction. And $P_\pi : \mathbb{S} \rightarrow \mathbb{B}$ is a predicate on everything.

COMMENT Some paradoxes derive from this theorem, see §3.2.12. For a constructive vision of \mathbb{S} , see Section §5. See also Subsection §6.2.

COROLLARY $\mathbb{P} \subset \mathbb{S}$ and $\mathbb{R} \subset \mathbb{S}$. Even $\mathbb{B} \subset \mathbb{S}$.

COMMENT If you are a teacher looking for a problem to ask in an exam, then your solution is a problem, so $\mathbb{P} \subset \mathbb{S}$ makes sense. And if you are a mathematician looking for an algorithm to resolve some kind of problems, then your solution is a resolution, so $\mathbb{R} \subset \mathbb{S}$ makes sense. There are many yes-or-no questions, so $\mathbb{B} \subset \mathbb{S}$ makes sense.

§3.2.2 NOTATION Let Σ_π be the (possibly infinite) set of all the solutions to problem π . So $\Sigma_\pi \subseteq \mathbb{S}$, or $\Sigma_\pi \in 2^{\mathbb{S}}$, and $\Sigma_\pi = \{s \mid P_\pi(s)\}$.

COMMENT A solution of the problem is any use of freedom that satisfies the condition, see Section §2, so s is a solution of problem π , if $P_\pi(s)$ stands.

COMMENT The condition of the problem π is the characteristic function of its set of solutions, that is, P_π is the characteristic function of Σ_π .

§3.2.3 LEMMA $\Sigma_{\pi \vee \rho} = \Sigma_{\pi} \cup \Sigma_{\rho}$, and $\Sigma_{\pi \wedge \rho} = \Sigma_{\pi} \cap \Sigma_{\rho}$, and $\Sigma_{\bar{\pi}} = \overline{\Sigma_{\pi}}$.

PROOF Just apply the definitions in §3.1.10:

$$\begin{aligned}\Sigma_{\pi \vee \rho} &= \{s \mid P_{\pi \vee \rho}(s)\} = \{s \mid P_{\pi}(s) \vee P_{\rho}(s)\} = \{s \mid s \in \Sigma_{\pi} \vee s \in \Sigma_{\rho}\} = \Sigma_{\pi} \cup \Sigma_{\rho}. \\ \Sigma_{\pi \wedge \rho} &= \{s \mid P_{\pi \wedge \rho}(s)\} = \{s \mid P_{\pi}(s) \wedge P_{\rho}(s)\} = \{s \mid s \in \Sigma_{\pi} \wedge s \in \Sigma_{\rho}\} = \Sigma_{\pi} \cap \Sigma_{\rho}. \\ \Sigma_{\bar{\pi}} &= \{s \mid P_{\bar{\pi}}(s)\} = \{s \mid \neg P_{\pi}(s)\} = \{s \mid s \notin \Sigma_{\pi}\} = \overline{\Sigma_{\pi}}. \diamond\end{aligned}$$

§3.2.4 LEMMA For a tautological problem, $x? P_{\tau}(x)$, everything is a solution, $\Sigma_{\tau} = \mathbb{S}$.

For a contradictory problem, $x? P_{\bar{\tau}}(x)$, nothing is a solution, $\Sigma_{\bar{\tau}} = \emptyset$.

PROOF $\Sigma_{\tau} = \{s \mid P_{\tau}(s)\} = \{s \mid \top\} = \mathbb{S}$. $\Sigma_{\bar{\tau}} = \{s \mid P_{\bar{\tau}}(s)\} = \{s \mid \perp\} = \{\} = \emptyset$. \diamond

§3.2.5 LEMMA $\Sigma_{\pi} \cup \Sigma_{\bar{\pi}} = \mathbb{S}$ and $\Sigma_{\pi} \cap \Sigma_{\bar{\pi}} = \emptyset$.

PROOF $\Sigma_{\pi} \cup \Sigma_{\bar{\pi}} = \{s \mid P_{\pi}(s)\} \cup \{s \mid \neg P_{\pi}(s)\} = \{s \mid P_{\pi}(s) \vee \neg P_{\pi}(s)\} = \{s \mid \top\} = \mathbb{S}$.

$$\Sigma_{\pi} \cap \Sigma_{\bar{\pi}} = \{s \mid P_{\pi}(s)\} \cap \{s \mid \neg P_{\pi}(s)\} = \{s \mid P_{\pi}(s) \wedge \neg P_{\pi}(s)\} = \{s \mid \perp\} = \emptyset. \diamond$$

§3.2.6 LEMMA The solutions of $\pi \wedge \rho$ are solutions of π and of ρ .

PROOF $\forall s \in \mathbb{S}; s \in \Sigma_{\pi \wedge \rho} \Leftrightarrow s \in \Sigma_{\pi} \cap \Sigma_{\rho} \Leftrightarrow s \in \Sigma_{\pi} \wedge s \in \Sigma_{\rho}$. \diamond

COMMENT The reader is free to explore this Boolean landscape, but here we will close with the following theorems.

§3.2.7 THEOREM $\langle 2^{\mathbb{S}}, \cup, \cap, -, \emptyset, \mathbb{S} \rangle$ is a Boolean algebra, where \emptyset is the neutral for \cup , and \mathbb{S} is the neutral for \cap .

PROOF The powerset of a set M , with the operations of union \cup , intersection \cap , and complement with respect to set M , noted \overline{Q} , is a typical example of a Boolean algebra. In detail, $\forall Q, R, S \in 2^{\mathbb{S}}$:

$$\begin{aligned}1o. (Q \cup R) \cup S &= Q \cup (R \cup S). & 1a. (Q \cap R) \cap S &= Q \cap (R \cap S). \\ 2o. Q \cup R &= R \cup Q. & 2a. Q \cap R &= R \cap Q. \\ 3o. Q \cup \emptyset &= Q. & 3a. Q \cap \mathbb{S} &= Q. \\ 4o. Q \cup \overline{Q} &= \mathbb{S}. & 4a. Q \cap \overline{Q} &= \emptyset. \\ 5o. Q \cup (R \cap S) &= (Q \cup R) \cap (Q \cup S). & 5a. Q \cap (R \cup S) &= (Q \cap R) \cup (Q \cap S). \diamond\end{aligned}$$

§3.2.8 THEOREM $\langle \mathbb{P}, \vee, \wedge, \neg, \bar{\tau}, \tau \rangle$ is isomorphic to $\langle 2^{\mathbb{S}}, \cup, \cap, -, \emptyset, \mathbb{S} \rangle$, that is, $\mathbb{P} \cong 2^{\mathbb{S}}$.

PROOF We define the bijection Σ that relates each problem π with the set of its solutions Σ_{π} : for every problem $\pi \in \mathbb{P}$ there is a set, the set of its solutions, $\Sigma_{\pi} \in 2^{\mathbb{S}}$, and for every set $S \in 2^{\mathbb{S}}$ there is a problem, $\pi_S = x? [x \in S]$, where $\pi_S \in \mathbb{P}$. Now, by Lemma §3.2.3, the bijection translates properly all three operations, $\vee \leftrightarrow \cup$, $\wedge \leftrightarrow \cap$, $\neg \leftrightarrow -$, and, by Lemma §3.2.4, also the two neutrals, $\bar{\tau} \leftrightarrow \emptyset$, $\tau \leftrightarrow \mathbb{S}$. \diamond

COMMENT We will call $\mathbb{P} \cong 2^{\mathbb{S}}$ the set isomorphism. That is, $\mathbb{P} \Leftrightarrow 2^{\mathbb{S}} : \pi \leftrightarrow \Sigma_{\pi}$.

COMMENT Using the set isomorphism, two problems are equal if they have the same solutions, that is, $\pi = \rho \Leftrightarrow \Sigma_{\pi} = \Sigma_{\rho}$.

§3.2.9 THEOREM The set of problems is equal to the powerset of the solutions, that is, $\mathbb{P} = 2^{\mathbb{S}}$.

PROOF The equality $\mathbb{P} = 2^{\mathbb{S}}$ derives directly from the set isomorphism $\mathbb{P} \cong 2^{\mathbb{S}}$, see §3.2.8, because no property was abstracted out. \diamond

§3.2.10 DEFINITION The set of singletons is: $\mathbb{S}^1 = \{S \in 2^{\mathbb{S}} \mid [|S| = 1]\}$.

PROPOSITION $\mathbb{S}^1 \subset 2^{\mathbb{S}}$, because $\forall S \in \mathbb{S}^1, S \in 2^{\mathbb{S}}$, but $\emptyset \in 2^{\mathbb{S}}$ and $\emptyset \notin \mathbb{S}^1$.

§3.2.11 DEFINITION The singleton isomorphism is the isomorphism between \mathbb{S} and \mathbb{S}^1 that relates each $s \in \mathbb{S}$ to the set $\{s\} \in \mathbb{S}^1$, and the converse. That is, $\mathbb{S} \cong \mathbb{S}^1$, and $\mathbb{S} \Leftrightarrow \mathbb{S}^1 : s \leftrightarrow \{s\}$.

COMMENT We can extend any operation on \mathbb{S} to \mathbb{S}^1 . For example, for any binary operation $*$ on \mathbb{S} , we define $\{a\} * \{b\} = \{a * b\}$.

COMMENT From the singleton isomorphism: $\mathbb{S} \cong \mathbb{S}^1 \subset 2^{\mathbb{S}}$.

§3.2.12 LEMMA The set of solutions \mathbb{S} is a proper subset of the set of problems \mathbb{P} , that is, $\mathbb{S} \subset \mathbb{P}$.

PROOF By the singleton isomorphism, see §3.2.11, $\mathbb{S} \cong \mathbb{S}^1$, and, by the set isomorphism, see §3.2.8, for each singleton there is a problem, so $\mathbb{S}^1 \subset \mathbb{P}$, and then $\mathbb{S} \cong \mathbb{S}^1 \subset \mathbb{P}$. \diamond

PARADOX We have both, $\mathbb{S} \subset \mathbb{P}$ and, by §3.2.1, $\mathbb{P} \subset \mathbb{S}$.

COMMENT If we only accept computable functions and computable sets, then $\mathbb{S}^* \not\subset \mathbb{P}^*$, see Subsection §6.2.

§3.2.13 DEFINITION A problem π is solved if a solution of π is known.

COMMENT To solve a problem, given the set of its solutions Σ_π , a choice function $f_c : 2^{\mathbb{S}} \setminus \emptyset \rightarrow \mathbb{S}$ is needed.

§3.2.14 DEFINITION A problem is unsolvable if $\Sigma_\pi = \{\} = \emptyset$, that is, if $|\Sigma_\pi| = 0$. A problem is solvable if $|\Sigma_\pi| > 0$.

COMMENT If a problem has not any solution, then it is unsolvable. If a problem has a solution, then it can be solved. A problem is solvable if it can be solved.

COMMENT Solved implies solvable, but not the converse: Solved \Rightarrow Solvable.

§3.3 Routines and Trials

§3.3.1 DEFINITION We will refer to the routine of problem π as R_π . The routine is the set of the solutions to the problem, a set that is known, see §2.3. Then $R_\pi = \Sigma_\pi$.

COMMENT The routine of problem π , R_π , is then, or an extensive definition of Σ_π , $\Sigma_\pi = \{s_1, \dots, s_n\}$, or a procedure \mathcal{P} that generates all problem π solutions and then halts. If the number of solutions is infinite, $|\Sigma_\pi| \geq \aleph_0$, then R_π has to be a procedure \mathcal{P} that keeps generating solutions forever.

§3.3.2 DEFINITION A trial on problem π over the set of possible solutions S , written $T_\pi(S)$, returns the set of those elements in S that satisfy the problem condition P_π , see §2.3. Then $T_\pi(S) = \{s \in S \mid P_\pi(s)\}$.

COMMENT Mathematically we will ignore the practical problem of governing the trial. Practically we will need a halt condition to truncate the calculations that are too long (or infinite), and some ordering on the tests to fit the execution of the tests to the available calculating machinery.

§3.3.3 DEFINITION To test if a possible solution $s \in S$ is a solution to problem π , is to replace the free variable with s . So, being $\pi = x? P_\pi(x)$, then to test if s is a solution is to calculate $P_\pi(s)$.

COMMENT Testing is a calculation $\mathbb{S} \rightarrow \mathbb{B}$.

§3.3.4 REMARK Replacing variables in expressions requires not confusing free with bound variables, nor bound with free variables.

COMMENT This means that the rule of β -conversion and the rules γ for substitution stand for testing. See [Curry & Feys \(1958\)](#), Section 3D for β -conversion, and Section 3E for substitution (the rules γ).

§3.3.5 THEOREM A trial on problem π over the set S is equal to the intersection of S with the set of the solutions Σ_π , that is, $T_\pi(S) = S \cap \Sigma_\pi$.

PROOF $T_\pi(S) = \{s \in S \mid P_\pi(s)\} = \{s \mid s \in S \wedge P_\pi(s)\} = \{s \mid s \in S \wedge s \in \Sigma_\pi\} = \{s \mid s \in S\} \cap \{s \mid s \in \Sigma_\pi\} = S \cap \Sigma_\pi$. \diamond

COROLLARY Any trial is a subset of the set of solutions, $T_\pi(S) \subseteq \Sigma_\pi$.

PROOF $T_\pi(S) = S \cap \Sigma_\pi \subseteq \Sigma_\pi$. \diamond

COROLLARY Any trial is a subset of the routine, that is, $T_\pi(S) \subseteq \Sigma_\pi = R_\pi$.

§3.3.6 LEMMA If S is a superset of Σ_π , then a trial on problem π over S is equal to Σ_π , and the converse, that is, $\Sigma_\pi \subseteq S \Leftrightarrow T_\pi(S) = \Sigma_\pi$.

PROOF $\Sigma_\pi \subseteq S \Leftrightarrow S \cap \Sigma_\pi = \Sigma_\pi \Leftrightarrow T_\pi(S) = \Sigma_\pi$, using Theorem §3.3.5. \diamond

COROLLARY If S is a superset of Σ_π , then a trial on problem π over S is equal to the routine of π , and the converse, that is, $\Sigma_\pi \subseteq S \Leftrightarrow T_\pi(S) = R_\pi$.

PROOF $\Sigma_\pi \subseteq S \Leftrightarrow S \cap \Sigma_\pi = \Sigma_\pi \Leftrightarrow T_\pi(S) = R_\pi$. \diamond

COROLLARY A trial on problem π over the whole \mathbb{S} is equal to Σ_π , that is, $T_\pi(\mathbb{S}) = \Sigma_\pi$.

PROOF Because $\Sigma_\pi \subseteq \mathbb{S}$. \diamond

COMMENT $T_\pi(\mathbb{S})$ is an exhaustive search.

§3.3.7 THEOREM The routine is a trial over all the solutions, that is, $R_\pi = T_\pi(\Sigma_\pi)$.

PROOF By Theorem §3.3.5, $T_\pi(\Sigma_\pi) = \Sigma_\pi \cap \Sigma_\pi = \Sigma_\pi = R_\pi$. \diamond

COMMENT $T_\pi(R_\pi) = T_\pi(\Sigma_\pi) = \Sigma_\pi = R_\pi$.

§3.4 Analogies

§3.4.1 DEFINITION If A is an analogy, and $\pi = x? P_\pi(x)$ is a problem, then $A\pi$ is another problem $A\pi = x? P_{A\pi}(x)$. That is, $A : \mathbb{P} \rightarrow \mathbb{P}$.

COMMENT So analogies transform a condition into a condition, P_π into $P_{A\pi}$ in this example.

COMMENT Taking advantage of problem decomposition, see §3.1.10, the result of an analogy, $A\pi$, can be a composition of problems that are easier to resolve than the original problem, π , see §2.3.

§3.4.2 DEFINITION If $\Sigma_\pi = \Sigma_{A\pi}$, then we say that the analogy is conservative.

COMMENT If an analogy is not conservative, then a function \mathcal{T}_A to translate $\Sigma_{A\pi}$ to Σ_π is required, because otherwise the analogy would be useless.

§3.4.3 NOTATION We will call function \mathcal{T}_A the translating function of analogy A .

$\mathcal{T}_A : 2^{\mathbb{S}} \rightarrow 2^{\mathbb{S}}$ and $\mathcal{T}_A(\Sigma_{A\pi}) = \Sigma_\pi$.

§3.4.4 LEMMA An analogy followed by another one is an analogy.

PROOF Because any analogy transforms a problem into a problem: $\mathbb{P} \rightarrow \mathbb{P}$. \diamond

COROLLARY Analogies can be chained.

§3.4.5 LEMMA Using only analogies we cannot resolve any problem.

PROOF Because using analogies we only get problems. \diamond

COMMENT While routines R and trials $T(S)$ are functions that return a set, $\mathbb{P} \rightarrow 2^{\mathbb{S}}$, analogies A are functions that return a function, $\mathbb{P} \rightarrow \mathbb{P}$.

§3.4.6 NOTATION We will write $A \circ T$ to express the composition of functions, where A is applied first and then T .

COMMENT $[A \circ T](x) = T(A(x))$. Diagram: $x \xrightarrow{A} A(x) \xrightarrow{T} T(A(x))$.

COMMENT If A_1 and A_2 are analogies, then $A_1 \circ A_2$ is also an analogy, by Lemma §3.4.4.

§3.4.7 DEFINITION To resolve a problem by analogy A is to compose $A \circ \mathfrak{R} \circ \mathcal{T}_A$, where \mathfrak{R} is any resolution, and \mathcal{T}_A is the translating function of A . Diagrams:

$$\pi \xrightarrow{A} A\pi \xrightarrow{\mathfrak{R}} \Sigma_{A\pi} \xrightarrow{\mathcal{T}_A} \Sigma_{\pi} \quad \text{or} \quad \mathbb{P} \xrightarrow{A} \mathbb{P} \xrightarrow{\mathfrak{R}} 2^{\mathbb{S}} \xrightarrow{\mathcal{T}_A} 2^{\mathbb{S}}.$$

COMMENT Analogy A is a translation from some original problem domain to some analogue problem domain. Then, by Lemma §3.4.5, we need a resolution \mathfrak{R} to resolve the analogue problem. And, finally, we need to translate the solutions back to the original domain.

§3.4.8 LEMMA The translating function of the composition $A \circ A'$ is $\mathcal{T}_{A'} \circ \mathcal{T}_A$.

PROOF If $\mathfrak{R} = A' \circ \mathfrak{R}' \circ \mathcal{T}_{A'}$ then we get $A \circ (A' \circ \mathfrak{R}' \circ \mathcal{T}_{A'}) \circ \mathcal{T}_A = A \circ A' \circ \mathfrak{R}' \circ \mathcal{T}_{A'} \circ \mathcal{T}_A = (A \circ A') \circ \mathfrak{R}' \circ (\mathcal{T}_{A'} \circ \mathcal{T}_A)$, because function composition is associative. Diagram:

$$\mathbb{P} \xrightarrow{A} \mathbb{P} \xrightarrow{A'} \mathbb{P} \xrightarrow{\mathfrak{R}'} 2^{\mathbb{S}} \xrightarrow{\mathcal{T}_{A'}} 2^{\mathbb{S}} \xrightarrow{\mathcal{T}_A} 2^{\mathbb{S}}. \quad \diamond$$

$\underbrace{\hspace{10em}}_{\mathfrak{R}}$

COROLLARY The translating function of the composition $A_1 \circ A_2 \dots \circ A_n$ is $\mathcal{T}_{A_n} \circ \dots \circ \mathcal{T}_{A_2} \circ \mathcal{T}_{A_1}$. That is: $\mathcal{T}_{A_1 \circ A_2 \dots \circ A_n} = \mathcal{T}_{A_n} \circ \dots \circ \mathcal{T}_{A_2} \circ \mathcal{T}_{A_1}$.

COMMENT This is how analogies can be chained.

§3.4.9 DEFINITION The identity function, written I , transforms anything into itself: $\forall x, I(x) = x$.

COMMENT The identity function I is an effectively calculable function, see §3.1.5. It is λ -definable; in λ -calculus, $I = (\lambda x.x)$.

COMMENT Identity I transforms π into π , $I(\pi) = \pi$, and P_{π} into P_{π} , $I(P_{\pi}) = P_{\pi}$.

COMMENT Identity I can work as an analogy: $I\pi = I(\pi) = \pi$.

§3.4.10 LEMMA The translating function of the identity analogy is the identity function: $\mathcal{T}_I = I$.

PROOF Because $I(\Sigma_{\pi}) = \Sigma_{\pi}$. Diagram: $\pi \xrightarrow{I} \pi \xrightarrow{\mathfrak{R}} \Sigma_{\pi} \xrightarrow{I} \Sigma_{\pi}$. \diamond

COMMENT The identity analogy is conservative, see §3.4.2.

§3.4.11 LEMMA The identity I followed by any function f , or any function f followed by identity I , is equal to the function: $\forall f, I \circ f = f = f \circ I$.

PROOF $\forall f, \forall x, [I \circ f](x) = f(I(x)) = f(x) = I(f(x)) = [f \circ I](x)$. \diamond

COMMENT $I \circ \mathfrak{R}(I\pi) \circ \mathcal{T}_I = I \circ \mathfrak{R}(\pi) \circ I = \mathfrak{R}(\pi)$.

§3.4.12 THEOREM $A \circ T_{A\pi}(S) \circ \mathcal{T}_A$, where A is an analogy, $T_{A\pi}(S)$ is a trial, and \mathcal{T}_A is the translating function of A , is the general form of a resolution.

PROOF If the analogy is the identity I , then the general form is reduced to $T_\pi(S)$, because $\mathcal{T}_I = I$, $I\pi = \pi$, so $I \circ T_{I\pi}(S) \circ I = T_\pi(S)$, which is a trial. By Theorem §3.3.7, a routine is a specific trial, $R_\pi = T_\pi(R_\pi)$, so $I \circ T_\pi(R_\pi) \circ I = T_\pi(R_\pi) = R_\pi$ reduces the general form to the routine. Resolving by analogy is, by definition, $A \circ \mathfrak{R} \circ \mathcal{T}_A$, and analogies can be chained, by Lemma §3.4.4, so a chain of analogies is an analogy, $A_1 \circ A_2 \circ \dots \circ A_n = A$, and by Lemma §3.4.8, $\mathcal{T}_A = \mathcal{T}_{A_1 \circ A_2 \circ \dots \circ A_n} = \mathcal{T}_{A_n} \circ \dots \circ \mathcal{T}_{A_2} \circ \mathcal{T}_{A_1}$. Then $A_1 \circ A_2 \circ \dots \circ A_n \circ T_{A\pi}(S) \circ \mathcal{T}_{A_n} \circ \dots \circ \mathcal{T}_{A_2} \circ \mathcal{T}_{A_1} = A \circ T_{A\pi}(S) \circ \mathcal{T}_A$. \diamond

SUMMARY There are three ways to resolve a problem: routine $R_\pi = I \circ T_\pi(R_\pi) \circ I$, trial $T_\pi(S) = I \circ T_\pi(S) \circ I$, and analogy $A_1 \circ \dots \circ A_n \circ T_{A\pi}(S) \circ \mathcal{T}_{A_n} \circ \dots \circ \mathcal{T}_{A_1} = A \circ T_{A\pi}(S) \circ \mathcal{T}_A$.

§3.5 Metaproblems

§3.5.1 DEFINITION A resolution $\mathfrak{R} : \mathbb{P} \rightarrow 2^{\mathbb{S}}$ is a valid resolution for a problem π if it finds all the solutions of problem π and then halts. In other words, \mathfrak{R} is a valid resolution for π if it satisfies two conditions: that $\mathfrak{R}(\pi)$ is effectively calculable, and that \mathfrak{R} fits problem π , that is, that $\mathfrak{R}(\pi) = \Sigma_\pi$.

COMMENT If Σ_π is infinite, $|\Sigma_\pi| \geq \aleph_0$, then a valid $\mathfrak{R}(\pi)$ does not halt, but it keeps building Σ_π forever.

§3.5.2 DEFINITION A problem π is resolved if a valid resolution for π is known.

COMMENT To solve a problem we have to find one solution, see §3.2.13. To resolve a problem we have to find all the solutions. To resolve a problem is to exhaust the problem.

§3.5.3 LEMMA Once a problem is resolved, we can thereafter resolve it by routine.

PROOF Once a problem is resolved, we know all of its solutions, Σ_π , and knowing Σ_π , we know its routine resolution, because $R_\pi = \Sigma_\pi$, see §3.3.1. \diamond

PROPOSITION If $\pi \wedge \rho$ is solvable, then by resolving $\pi \wedge \rho$ both π and ρ are solved.

§3.5.4 DEFINITION A problem is resolvable if there is a valid resolution for the problem, see §3.5.1, that is, if there is a resolution \mathfrak{R} such that $\mathfrak{R}(\pi)$ is effectively calculable, and $\mathfrak{R}(\pi) = \Sigma_\pi$. Otherwise, the problem is unresolvable.

COMMENT A problem is resolvable if it can be resolved.

COMMENT Resolved implies resolvable, but not the converse: Resolved \Rightarrow Resolvable.

§3.5.5 DEFINITION For any Boolean-valued function $P : \mathbb{S} \rightarrow \mathbb{B}$, we define the function $\check{P} : \mathbb{B} \rightarrow 2^{\mathbb{S}}$, called the inverse of condition P , as follows:

$$\begin{aligned}\check{P}(\top) &= \{x \mid [P(x) = \top]\}, \\ \check{P}(\perp) &= \{x \mid [P(x) = \perp]\}.\end{aligned}$$

§3.5.6 LEMMA If $P_\pi(x)$ is the condition of a problem π , then $\check{P}_\pi(\top) = \Sigma_\pi$ and $\check{P}_\pi(\perp) = \overline{\Sigma_\pi} = \Sigma_{\bar{\pi}}$.

PROOF Because $\check{P}_\pi(\top) = \{x \mid [P_\pi(x) = \top]\} = \{x \mid P_\pi(x)\} = \Sigma_\pi$, and $\check{P}_\pi(\perp) = \{x \mid [P_\pi(x) = \perp]\} = \{x \mid \neg P_\pi(x)\} = \overline{\Sigma_\pi} = \Sigma_{\bar{\pi}}$, by Lemma §3.2.3. \diamond

§3.5.7 THEOREM The inverse of the condition of a problem, provided it is an effectively calculable function, resolves the problem and its complementary by routine.

PROOF By §3.5.6 and §3.3.1, $\check{P}_\pi(\top) = \Sigma_\pi = R_\pi$, then $\check{P}_\pi(\top)$ is the routine resolution of π , if $\check{P}_\pi(\top)$ is effectively calculable, see §3.1.5. And if $\check{P}_\pi(\perp)$ is effectively calculable, then it resolves the complementary problem by routine, $\check{P}_\pi(\perp) = \Sigma_{\bar{\pi}} = R_{\bar{\pi}}$. \diamond

COMMENT It is a nice theorem, but how can we find the inverse of a condition?

§3.5.8 DEFINITION The metaproblem of a problem, written $\Pi\pi$, is the problem of finding the valid resolutions for problem π . In other words, if $\pi = x? P_\pi(x)$, then $\Pi\pi = \mathfrak{R}?$ [$\mathfrak{R}(\pi) = \Sigma_\pi$].

COMMENT The solutions of the metaproblems are the resolutions, $\Pi\mathbb{S} = \mathbb{R}$.

COMMENT The condition of the metaproblem, $P_{\Pi\pi}$, is $[\mathfrak{R}(\pi) = \Sigma_\pi]$, that is, $P_{\Pi\pi}(\mathfrak{R}) = [\mathfrak{R}(\pi) = \Sigma_\pi]$, or using an α -conversion, $P_{\Pi\pi}(x) = [x(\pi) = \Sigma_\pi]$.

§3.5.9 LEMMA A metaproblem is a problem, that is, $\Pi\mathbb{P} \subset \mathbb{P}$.

PROOF Because $\Pi\pi = x? P_{\Pi\pi}(x)$, but some problems are not metaproblems. \diamond

COMMENT A metaproblem is a problem because it has its two ingredients: there are several ways to resolve a problem, so there is freedom, but only the valid resolutions resolve the problem, so there is a condition.

§3.5.10 DEFINITION The metacondition P_Π is $P_\Pi(p, r) = [r(p) = \Sigma_p]$, for any problem $p \in \mathbb{P}$, and for any resolution $r \in \mathbb{R}$.

COMMENT Using another α -conversion, $P_\Pi(\pi, x) = [x(\pi) = \Sigma_\pi] = P_{\Pi\pi}(x)$.

COMMENT $\Pi\pi = x? P_\Pi(\pi, x)$.

§3.5.11 DEFINITION Metaresolving is resolving the metaproblem to resolve the problem.

COMMENT Metaresolving is a kind of analogy. Diagram:

$$\pi \xrightarrow{\Pi} \Pi\pi \xrightarrow{\Pi\mathfrak{R}} \Sigma_{\Pi\pi} = \{ \mathfrak{R} \mid [\mathfrak{R}(\pi) = \Sigma_\pi] \} \xrightarrow{f_c} \mathfrak{R}_c \xrightarrow{(\pi)} \mathfrak{R}_c(\pi) = \Sigma_\pi.$$

Function f_c is a choice function, and the last calculation, noted (π) , means to apply π as the argument, not as the function. If you only metasolve, then you don't need to choose. In any case, the translating function of metaresolving is $\mathcal{T}_\Pi = f_c \circ (\pi)$. Then we can draw the following diagrams:

$$\pi \xrightarrow{\Pi} \Pi\pi \xrightarrow{\Pi\mathfrak{R}} \Sigma_{\Pi\pi} \xrightarrow{\mathcal{T}_\Pi} \Sigma_\pi \quad \text{or} \quad \mathbb{P} \xrightarrow{\Pi} \Pi\mathbb{P} \xrightarrow{\Pi\mathfrak{R}} 2^{\Pi\mathbb{S}} = 2^{\mathbb{R}} \xrightarrow{\mathcal{T}_\Pi} 2^{\mathbb{S}}.$$

§3.5.12 LEMMA The metaproblem $\Pi\pi$ of some problem π is solvable if, and only if, the problem π is resolvable, that is, $\Pi\pi$ is solvable $\Leftrightarrow \pi$ is resolvable.

PROOF If $\Pi\pi$ is solvable, then there is a solution to it, see §3.2.14, and that solution is a valid resolution for π , see §3.5.8, and then π is resolvable, see §3.5.4. If π is resolvable, then there is a valid resolution for it, see §3.5.4, and that resolution is a solution of its metaproblem $\Pi\pi$, see §3.5.8, and then $\Pi\pi$ is solvable, see §3.2.14. \diamond

COROLLARY To solve the metaproblem $\Pi\pi$ of problem π is to resolve problem π .

PROOF Because to resolve problem π is to find a valid resolution for π , see §3.5.2, and to solve the metaproblem $\Pi\pi$ is to find a solution to $\Pi\pi$, see §3.2.13, which is also to find a valid resolution for π , see §3.5.8. \diamond

COMMENT And again, $\mathbb{R} = \Pi\mathbb{S}$.

§3.5.13 LEMMA The set of the valid resolutions for problem π is the routine resolution of its metaproblem $\Pi\pi$, that is, $\{\mathfrak{R} \mid [\mathfrak{R}(\pi) = \Sigma_\pi]\} = R_{\Pi\pi}$.

PROOF $R_{\Pi\pi} = \Sigma_{\Pi\pi}$, by the definition of routine, see §3.3.1.

And $\Sigma_{\Pi\pi} = \{\mathfrak{R} \mid [\mathfrak{R}(\pi) = \Sigma_\pi]\}$, by the definition of $\Pi\pi$, see §3.5.8. \diamond

§3.6 Resolution Typology

§3.6.1 DEFINITION The metaⁿ-metaproblem of π , $\Pi^n\Pi\pi$, is (the metaproblem of)ⁿ the metaproblem of π , where $n \in \mathbb{N}$.

SPECIAL CASE The meta-metaproblem of π , $\Pi\Pi\pi = \Pi^1\Pi\pi$, is the metaproblem of the metaproblem of π .

EXAMPLES $\Pi^0\Pi\pi = \Pi\pi$. $\Pi^1\Pi\pi = \Pi\Pi\pi = \Pi^2\pi$. $\Pi^2\Pi\pi = \Pi\Pi\Pi\pi = \Pi^3\pi$.

COMMENT From $\Pi\mathbb{S} = \mathbb{R}$, we get $\Pi\Pi\mathbb{S} = \Pi\mathbb{R}$ and $\Pi^n\Pi\mathbb{S} = \Pi^n\mathbb{R}$.

COMMENT The condition of the metaⁿ-metaproblem of π , $P_{\Pi^n\Pi\pi}$, where $n \in \mathbb{N}$, is:

$$P_{\Pi^n\Pi\pi}(x) = [x(\Pi^n\pi) = \Sigma_{\Pi^n\pi}].$$

EXAMPLES $P_{\Pi^0\Pi\pi}(x) = [x(\Pi^0\pi) = \Sigma_{\Pi^0\pi}] = [x(\pi) = \Sigma_\pi] = P_{\Pi\pi}(x)$.

$$P_{\Pi^1\Pi\pi}(x) = [x(\Pi^1\pi) = \Sigma_{\Pi^1\pi}] = [x(\Pi\pi) = \Sigma_{\Pi\pi}] = P_{\Pi\Pi\pi}(x).$$

§3.6.2 LEMMA A metaⁿ-metaproblem is a problem, where $n \in \mathbb{N}$.

PROOF If $n > 0$, then $\Pi^n\Pi\pi = x? P_{\Pi^n\Pi\pi}(x)$. For $n = 0$, see §3.5.9. \diamond

COROLLARY $\bigcup_{n \in \mathbb{N}} \Pi^n\Pi\mathbb{P} \subset \mathbb{P}$.

§3.6.3 DEFINITION The metaⁿ-metacondition $P_{\Pi^n\Pi}$, with $n \in \mathbb{N}$, $p \in \mathbb{P}$, and $r \in \mathbb{R}$ is:

$$P_{\Pi^n\Pi}(p, r) = [r(\Pi^n p) = \Sigma_{\Pi^n p}].$$

COMMENT Using an α -conversion, $P_{\Pi^n\Pi}(\pi, x) = [x(\Pi^n\pi) = \Sigma_{\Pi^n\pi}] = P_{\Pi^n\Pi\pi}(x)$.

EXAMPLE $P_{\Pi^1\Pi}(\pi, x) = P_{\Pi\Pi}(\pi, x) = [x(\Pi\pi) = \Sigma_{\Pi\pi}] = [x(\Pi^1\pi) = \Sigma_{\Pi^1\pi}]$.

§3.6.4 LEMMA $P_{\Pi^n\Pi}(\pi, x) = P_{\Pi}(\Pi^n\pi, x)$, where $n \in \mathbb{N}$.

PROOF By §3.5.10, $P_{\Pi}(\Pi^n\pi, x) = [x(\Pi^n\pi) = \Sigma_{\Pi^n\pi}] = P_{\Pi^n\Pi}(\pi, x)$. \diamond

SPECIAL CASE $P_{\Pi\Pi}(\pi, x) = P_{\Pi}(\Pi\pi, x)$.

COMMENT The meta-metacondition is the metacondition of the metaproblem.

§3.6.5 LEMMA A metaⁿ-metaproblem is a metaproblem, where $n \in \mathbb{N}$.

PROOF If $n > 0$, $\Pi^n\Pi\pi = x? P_{\Pi^n\Pi}(\pi, x)$, and $\Pi^0\Pi\pi = \Pi\pi = x? P_{\Pi}(\pi, x)$. \diamond

COROLLARY $\bigcup_{n \in \mathbb{N}} \Pi^n\Pi\mathbb{P} = \Pi\mathbb{P}$.

§3.6.6 LEMMA We have the following infinite series of mathematical objects:

$$\mathbb{S}, \mathbb{P} = 2^{\mathbb{S}}, \mathbb{R} = \Pi\mathbb{S} = 2^{\mathbb{S}} \rightarrow 2^{\mathbb{S}}, \Pi\mathbb{P} = 2^{2^{\mathbb{S}} \rightarrow 2^{\mathbb{S}}}, \Pi\mathbb{R} = \Pi\Pi\mathbb{S} = 2^{2^{\mathbb{S}} \rightarrow 2^{\mathbb{S}}} \rightarrow 2^{2^{\mathbb{S}} \rightarrow 2^{\mathbb{S}}}, \dots$$

PROOF $\mathbb{P} = \mathbb{S} \rightarrow \mathbb{B} = 2^{\mathbb{S}}$, by Theorems §3.1.8 and §3.2.9.

$$\Pi\mathbb{S} = \mathbb{R}, \text{ by the metaproblem definition, see §3.5.8, and } \mathbb{R} = \mathbb{P} \rightarrow 2^{\mathbb{S}} = 2^{\mathbb{S}} \rightarrow 2^{\mathbb{S}}.$$

$$\Pi\mathbb{P} = \Pi\mathbb{S} \rightarrow \mathbb{B} = \mathbb{R} \rightarrow \mathbb{B} = 2^{\mathbb{R}} = 2^{2^{\mathbb{S}} \rightarrow 2^{\mathbb{S}}}.$$

$$\Pi\mathbb{R} = \Pi\mathbb{P} \rightarrow 2^{\Pi\mathbb{S}} = 2^{\mathbb{R}} \rightarrow 2^{\mathbb{R}} = 2^{2^{\mathbb{S}} \rightarrow 2^{\mathbb{S}}} \rightarrow 2^{2^{\mathbb{S}} \rightarrow 2^{\mathbb{S}}}.$$

And so on. \diamond

§3.6.7 THEOREM There is only one level of problem meta-ness.

PROOF By Lemma §3.6.5, because every metaⁿ-metaproblem is a metaproblem, and every metaproblem is a meta⁰-metaproblem, so $\bigcup_{n \in \mathbb{N}} \Pi^n\Pi\mathbb{P} = \Pi\mathbb{P} \subset \mathbb{P}$. \diamond

COMMENT While a problem condition is any predicate, $P(x)$, a metaproblem condition is a specific kind of predicate, namely, $P_{\Pi}(p, r) = [r(p) = \Sigma_p]$. And any metaⁿ-metaproblem condition, $P_{\Pi^n\Pi}$, is the same specific predicate P_{Π} , see §3.6.4.

COMMENT We are assuming that functions are free to take functions as arguments. See that, in predicate $P_{\Pi}(p, r) = [r(p) = \Sigma_p]$, argument r is a function in $\Pi^n\mathbb{R}$ that takes $p \in \Pi^n\mathbb{P}$ as argument. Therefore, the theorem holds unconditionally for λ -definable functions, including predicates, see §4.2.9. And then, under Church's thesis, see §4.3.1, the theorem is true for effectively calculable functions, and in particular, it is true for expressible and for resolvable problems, see §3.1.6 and §3.5.4.

§3.6.8 THEOREM There are five types of resolution.

PROOF From Theorem §3.4.12 we get three types for the resolution of problems: R_{π} , $T_{\pi}(S)$, and $A \circ T_{A\pi}(S) \circ \mathcal{T}_A$. This shows that there are several ways of resolving, so choosing a resolution that find solutions to the original problem π is another problem, the metaproblem $\Pi\pi$, see §3.5.8. Then we should get another three for the resolution of the metaproblem, but, by §3.5.13, the set of the resolutions of a problem is the routine resolution of its metaproblem, so we only add two more for the metaproblem: $T_{\Pi\pi}(R)$, and $\mathcal{A} \circ T_{\mathcal{A}\Pi\pi}(R) \circ \mathcal{T}_A$. Finally, by §3.6.7, we do not need to go deeper into metaⁿ-metaproblems. \diamond

COMMENT We will call them: routine R_{π} , trial $T_{\pi}(S)$, analogy $A \circ T_{A\pi}(S) \circ \mathcal{T}_A$, meta-trial $T_{\Pi\pi}(R)$, and meta-analogy $\mathcal{A} \circ T_{\mathcal{A}\Pi\pi}(R) \circ \mathcal{T}_A$. The first three can also be called meta-routines.

§3.6.9 REMARK The diagram for the meta-trial, or trial of the metaproblem, is:

$$\pi \xrightarrow{\Pi} \Pi\pi \xrightarrow{T_{\Pi\pi}(R)} \Sigma_{\Pi\pi} \xrightarrow{\mathcal{T}_{\Pi}} \Sigma_{\pi} .$$

And the diagram for the meta-analogy, or analogy of the metaproblem, is:

$$\pi \xrightarrow{\Pi} \Pi\pi \xrightarrow{\mathcal{A}} \mathcal{A}\Pi\pi \xrightarrow{T_{\mathcal{A}\Pi\pi}(R)} \Sigma_{\mathcal{A}\Pi\pi} \xrightarrow{\mathcal{T}_{\mathcal{A}}} \Sigma_{\Pi\pi} \xrightarrow{\mathcal{T}_{\Pi}} \Sigma_{\pi} .$$

See that $\mathcal{A} : \Pi\mathbb{P} \rightarrow \Pi\mathbb{P} = 2^{2^{\mathbb{S}} \rightarrow 2^{\mathbb{S}}} \rightarrow 2^{2^{\mathbb{S}} \rightarrow 2^{\mathbb{S}}}$ and $\mathcal{T}_{\mathcal{A}} : 2^{\Pi\mathbb{S}} \rightarrow 2^{\Pi\mathbb{S}} = 2^{2^{\mathbb{S}} \rightarrow 2^{\mathbb{S}}} \rightarrow 2^{2^{\mathbb{S}} \rightarrow 2^{\mathbb{S}}}$, using §3.6.6. Both are functions taking sets of functions on sets to sets and returning sets of functions on sets to sets.

§4 Computers

§4.1 Turing Machine

§4.1.1 DEFINITION A computation is any manipulation of a string of symbols, irrespective of the symbols meanings, but according to a finite set of well-defined rules.

COMMENT Computing is any mechanical transformation of a string of symbols.

§4.1.2 DEFINITION A computing device, or computer, is any mechanism that can perform computations.

COMMENT The prototype of computing device is a Turing machine, see Turing (1936).

§4.1.3 NOTATION The Turing machine has two parts: the processor \mathcal{P} , which is a finite state automaton, and an infinite tape, $\langle \rangle$, which in any moment contains only a finite number of symbols.

COMMENT In the case of a processor of a Turing machine, the output alphabet O , that is, the finite set of output symbols, has to be: $O = I^+ \times \{l, h, r\}$, where I is the finite not empty input alphabet, $I^+ = I \cup \{b\}$, where $b \notin I$ is *blank*, and l , h , and r mean *left*, *halt*, and *right*. Then its transition function is $\mathcal{T} : S \times I^+ \rightarrow S \times I^+ \times \{l, h, r\}$, where S is the finite set of internal states. And the strings that the Turing machine transforms are sequences of symbols taken from set I .

§4.1.4 NOTATION We will refer to the set of Turing machines as \mathfrak{T} . We will refer to the set of the strings of symbols as \mathfrak{E} .

COMMENT Because all Turing machines tapes are equal, the processor defines the Turing machine, and therefore we will refer to the Turing machine with processor \mathcal{P} as the Turing machine \mathcal{P} , and then $\mathcal{P} \in \mathfrak{T}$. We will refer to the string of symbols written on the tape as the expression $\mathfrak{e} \in \mathfrak{E}$.

§4.1.5 LEMMA The set of expressions is countable, that is, $|\mathfrak{E}| = |\mathbb{N}| = \aleph_0$.

PROOF Let I be any finite alphabet, and s its cardinality, that is, s is the number of symbols, $s = |I| > 0$. We write I^n the set of strings of length n , so $|I^n| = s^n$. Then $\mathfrak{E} = \bigcup_{n \in \mathbb{N}} I^n$, and we can define a bijection between \mathfrak{E} and \mathbb{N} this way: it maps the empty string in I^0 to 0, it maps the s strings in I^1 to the next s numbers, it maps the s^2 strings in I^2 to the next s^2 numbers, and so on. Note that ordering the symbols in I , we can order alphabetically the strings in each I^n . \diamond

COMMENT Most real numbers are not expressible. See [Turing \(1936\)](#) §10 for details; but, for example, transcendental numbers π and e are computable, page 256.

§4.1.6 NOTATION We will use the notation $\mathcal{P}\langle \mathfrak{e} \rangle \hookrightarrow \mathfrak{r}$ to indicate that, if we write the expression $\mathfrak{e} \in \mathfrak{E}$ on the tape of the Turing machine with processor \mathcal{P} and we leave it running, then when it halts we will find the expression $\mathfrak{r} \in \mathfrak{E}$ on the tape. If, on the contrary, the Turing machine \mathcal{P} does not halt when we write the expression \mathfrak{w} , then we would say that \mathfrak{w} is a paradox in \mathcal{P} , and we would indicate this as follows: $\mathcal{P}\langle \mathfrak{w} \rangle \hookrightarrow \infty$.

§4.1.7 DEFINITION $\mathfrak{E}^+ = \mathfrak{E} \cup \{\infty\}$.

COMMENT Some computations do not halt, so we need ∞ to refer to them. Note that $\infty \notin \mathfrak{E}$, but $\infty \in \mathfrak{E}^+$. So $\mathfrak{E} \subset \mathfrak{E}^+$.

§4.1.8 DEFINITION For each Turing machine $\mathcal{P} \in \mathfrak{T}$ we define a function $F_{\mathcal{P}} : \mathfrak{E} \rightarrow \mathfrak{E}^+$, this way:

$$F_{\mathcal{P}}(\mathfrak{e}) = \begin{cases} \mathfrak{r} & \text{if } \mathcal{P}\langle \mathfrak{e} \rangle \hookrightarrow \mathfrak{r} \\ \infty & \text{if } \mathcal{P}\langle \mathfrak{e} \rangle \hookrightarrow \infty \end{cases}.$$

COMMENT If $\forall \mathfrak{e} \in \mathfrak{E}, F_{\mathcal{P}}(\mathfrak{e}) = F_{\mathcal{Q}}(\mathfrak{e})$, then we say that Turing machines \mathcal{P} and \mathcal{Q} are behaviorally equivalent, $\mathcal{P} \equiv_F \mathcal{Q}$, or that \mathcal{P} and \mathcal{Q} implement the same function.

§4.1.9 DEFINITION We say that a function is computable if there is a Turing machine that implements the function.

§4.1.10 LEMMA For each Turing machine we can define a unique finite string of symbols, that is, $\exists \mathbf{c} : \mathfrak{T} \rightarrow \mathfrak{E}$ such that $\mathcal{P} = \mathcal{Q} \Leftrightarrow \mathbf{c}(\mathcal{P}) = \mathbf{c}(\mathcal{Q})$.

PROOF Proved by Turing (1936), §5. Turing machines are defined by their processors, which are finite state automata. And every finite state automaton is defined by the table that describes its transition function \mathcal{T} in full, which is a finite table of expressions referring to internal states, input symbols, and output symbols. A table can be converted to a string just using an additional symbol for the end of line, and another symbol for the end of cell. To assure uniqueness, we have to impose some order on the lines and on the cells. \diamond

COMMENT $\mathbf{c}(\mathcal{P}) \in \mathfrak{E}$ is the string of symbols that represents the Turing machine $\mathcal{P} \in \mathfrak{T}$.

§4.1.11 NOTATION We will refer to $\mathbf{p} = \mathbf{c}(\mathcal{P})$ as a program, and to the set of programs as \mathfrak{P} . The set of programs is a proper subset of the set of expressions, $\mathfrak{P} \subset \mathfrak{E}$.

§4.1.12 DEFINITION The program isomorphism is the natural isomorphism that relates each Turing machine $\mathcal{P} \in \mathfrak{T}$ with the expression describing it, $\mathbf{c}(\mathcal{P}) = \mathbf{p} \in \mathfrak{P}$. That is, $\mathfrak{T} \Leftrightarrow \mathfrak{P} : \mathcal{P} \leftrightarrow \mathbf{c}(\mathcal{P})$.

COMMENT Now, $\mathfrak{T} \cong \mathfrak{P} \subset \mathfrak{E}$.

§4.1.13 LEMMA The set of Turing machines is countable, that is, $|\mathfrak{T}| = |\mathbb{N}| = \aleph_0$.

PROOF Proved by Turing (1936), §5. Using the program isomorphism, see §4.1.12, we order the Turing machines according to its corresponding program $\mathbf{p} = \mathbf{c}(\mathcal{P})$. We can order the programs, because they are finite strings of symbols, for example first by length, and then those of a given length by some kind of alphabetical order. Once ordered, we can assign a natural number to each one. \diamond

§4.1.14 THEOREM All computing sets are countable, that is, $|\mathfrak{T}| = |\mathfrak{E}| = \aleph_0$.

PROOF By Lemmas §4.1.5 and §4.1.13. \diamond

COMMENT All computing is about countable sets. Computing is counting.

§4.2 Turing Completeness

§4.2.1 THEOREM There is a Turing machine, called universal Turing machine, \mathcal{U} , that can compute anything that any Turing machine can compute. That is:

$$\exists \mathcal{U} \in \mathfrak{T} \mid \forall \mathcal{P} \in \mathfrak{T}, \forall \mathfrak{d} \in \mathfrak{E}, \quad \mathcal{U}\langle \mathbf{c}(\mathcal{P}) \mathfrak{d} \rangle = \mathcal{P}\langle \mathfrak{d} \rangle.$$

PROOF Proved by Turing (1936), §6 and §7. \diamond

COMMENT The equality means that if $\mathcal{P}\langle \mathfrak{d} \rangle \hookrightarrow \mathfrak{r}$ then $\mathcal{U}\langle \mathbf{c}(\mathcal{P}) \mathfrak{d} \rangle \hookrightarrow \mathfrak{r}$, and the converse, and also that if $\mathcal{P}\langle \mathfrak{d} \rangle \hookrightarrow \infty$ then $\mathcal{U}\langle \mathbf{c}(\mathcal{P}) \mathfrak{d} \rangle \hookrightarrow \infty$, and the converse. That is, $\mathcal{U}\langle \mathbf{c}(\mathcal{P}) \rangle \equiv_F \mathcal{P}$. To complete the definition, if $\mathfrak{e} \notin \mathfrak{P}$, then $\mathcal{U}\langle \mathfrak{e} \mathfrak{d} \rangle \hookrightarrow \mathfrak{e} \mathfrak{d}$.

§4.2.2 NOTATION We will refer to the set of universal Turing machines as \mathfrak{U} .

COMMENT The set of universal Turing machines is a proper subset of the set of Turing machines, $\mathfrak{U} \subset \mathfrak{T}$.

§4.2.3 LEMMA For each universal Turing machine \mathcal{U} there is a universal program \mathbf{u} .

PROOF Universal Turing machines are Turing machines, and $\mathbf{u} = \mathbf{c}(\mathcal{U})$. Then, by the program isomorphism, see §4.1.12, $\mathbf{u} = \mathcal{U}$. \diamond

COMMENT Given $\mathbf{u} = \mathbf{c}(\mathcal{U})$ and $\mathbf{p} = \mathbf{c}(\mathcal{P})$, then $\mathcal{U}\langle \mathbf{p} \mathfrak{d} \rangle = \mathcal{P}\langle \mathfrak{d} \rangle$ and $\mathcal{U}\langle \mathbf{u} \mathbf{p} \mathfrak{d} \rangle = \mathcal{U}\langle \mathbf{p} \mathfrak{d} \rangle$, so \mathbf{u} is the identity for programs, and $\mathcal{U}\langle \mathbf{u} \mathbf{p} \mathfrak{d} \rangle = \mathcal{U}\langle \mathbf{u} \mathbf{p} \mathfrak{d} \rangle = \mathcal{U}\langle \mathbf{p} \mathfrak{d} \rangle = \mathcal{P}\langle \mathfrak{d} \rangle$.

§4.2.4 DEFINITION The terminating condition $P_\sigma : \mathfrak{T} \rightarrow \mathbb{B}$ is:

$$P_\sigma(\mathcal{P}) = \begin{cases} \perp & \text{if } \exists \mathfrak{w} \in \mathfrak{E}, \mathcal{P}\langle \mathfrak{w} \rangle \hookrightarrow \infty \\ \top & \text{otherwise.} \end{cases}$$

COMMENT A terminating Turing machine always halts. There are not paradoxes in a terminating Turing machine. While Turing machines implement partial functions, $\mathfrak{E} \rightarrow \mathfrak{E}^+$, see §4.1.8, terminating Turing machines implement total functions, $\mathfrak{E} \rightarrow \mathfrak{E}$.

§4.2.5 DEFINITION The terminating problem is $\sigma = p? P_\sigma(p)$.

The non-terminating problem is $\bar{\sigma} = p? \neg P_\sigma(p)$.

COMMENT The terminating problem follows from the condition isomorphism of problems, see §3.1.7, applied to the terminating condition P_σ .

The non-terminating problem is derived from the terminating one by negation, see §3.1.10.

COMMENT The set of terminating Turing machines is Σ_σ ,
and the set of non-terminating Turing machines is $\Sigma_{\bar{\sigma}}$.

PROPOSITION Σ_σ and $\Sigma_{\bar{\sigma}}$ are a partition of \mathfrak{T} , because $\Sigma_\sigma \cap \Sigma_{\bar{\sigma}} = \emptyset$ and $\Sigma_\sigma \cup \Sigma_{\bar{\sigma}} = \mathfrak{T}$.

§4.2.6 DEFINITION We will call $\mathfrak{a} = \mathfrak{c}(\mathcal{P}_\sigma) \in \mathfrak{E}$, where $\mathcal{P}_\sigma \in \Sigma_\sigma$, an algorithm.

COMMENT $\forall \mathfrak{d}, \mathcal{P}_\sigma \langle \mathfrak{d} \rangle \hookrightarrow \mathfrak{r} \neq \infty \Leftrightarrow \forall \mathfrak{d}, \mathcal{U} \langle \mathfrak{a} \mathfrak{d} \rangle \hookrightarrow \mathfrak{r} \neq \infty$.

COMMENT An algorithm is the expression of a computation that always halts.

NOTATION We will refer to the set of algorithms as \mathfrak{A} .

COMMENT $\mathfrak{A} \subset \mathfrak{P} \subset \mathfrak{E}$.

§4.2.7 LEMMA Universal Turing machines are non-terminating, that is, $\mathfrak{U} \subset \Sigma_{\bar{\sigma}} \subset \mathfrak{T}$.

PROOF Because there are paradoxes in some Turing machines. For example, for Turing machine \mathcal{W} , that has not any h (*halt*) in its transition table, every expression is a paradox. That is, $\exists \mathcal{P} \in \mathfrak{T}, \exists \mathfrak{w} \in \mathfrak{E}, \mathcal{P}\langle \mathfrak{w} \rangle \hookrightarrow \infty \Rightarrow \forall \mathcal{U} \in \mathfrak{U}, \mathcal{U}\langle \mathfrak{c}(\mathcal{P}) \mathfrak{w} \rangle \hookrightarrow \infty$. \diamond

COMMENT If expression \mathfrak{w} is a paradox in \mathcal{P} , then expression $\mathfrak{c}(\mathcal{P}) \mathfrak{w}$ is a paradox in \mathcal{U} . Then, $\mathcal{U} \in \Sigma_{\bar{\sigma}}$.

§4.2.8 DEFINITION A computing device is Turing complete if it can compute whatever any Turing machine can compute. We will call every Turing complete device a universal computer.

COMMENT The prototype of universal computer is a universal Turing machine, \mathcal{U} .

COMMENT The Turing machine, as it was presented by Turing (1936), models the calculations done by a person. This means that we can compute whatever any Turing machine can compute provided we have enough time and memory, and therefore we are Turing complete provided we have enough time and memory.

§4.2.9 THEOREM All universal computers are equivalent.

PROOF Gödel and Herbrand recursiveness, Church λ -definability, and Turing computability are equivalent, because Kleene (1936) showed that every recursive function is λ -definable, and the converse, and then Turing (1937) showed that every λ -definable function is computable, and that every computable function is recursive. \diamond

COMMENT A universal Turing machine is equivalent to a λ -calculus interpreter, where a λ -calculus interpreter is a device that can perform any λ -calculus reduction. A universal Turing machine is equivalent to a mathematician calculating formally, and without errors, any recursive function.

COMMENT The universal Turing machine, the λ -calculus interpreter, and the mathematician, who is a person, are equal in computing power. And all of them are Turing complete.

§4.2.10 PROVISIO Whenever we apply a general statement to a finite universal computing device, we should add a cautious ‘provided it has enough time and memory’.

COMMENT Although the finite universal computer can perform each and every step of the computation exactly the same as the unrestricted universal computer, the finite universal computer could meet some limitations of time or memory that would prevent it to complete the computation. In that case, the same finite universal computer, provided with some additional time and some more memory, would perform some more computing steps exactly the same as the unrestricted universal computer. This extension procedure can be repeated as desired to close the gap between the finite and the unrestricted universal computer.

COMMENT We will understand that the proviso ‘provided it has enough time and memory’ is implicitly stated whenever we refer to a finite universal computing device.

§4.2.11 CONVENTION Because all universal computers are equivalent, we can use any of them, let us call the one used \mathcal{U} , and then drop every \mathcal{U} from the formulas, and just examine expressions, that is, elements in \mathfrak{E} . In case we need to note a non-halting computation, we will use ∞ .

COMMENT Using the convention is as if we were always looking inside the tape of \mathcal{U} . Given a universal computer, \mathcal{U} , computing is about expressions manipulating expressions.

EXAMPLE Formula $\mathcal{U}\langle c(\mathcal{P}) \mathfrak{d} \rangle \hookrightarrow \mathfrak{r}$ is reduced to $\langle c(\mathcal{P}) \mathfrak{d} \rangle \hookrightarrow \mathfrak{r}$, and even to $\langle \mathfrak{p} \mathfrak{d} \rangle \hookrightarrow \mathfrak{r}$, using the rewriting rule: $\forall \mathcal{P} \in \mathfrak{T}, c(\mathcal{P}) = \mathfrak{p}$. If the universal computer is a λ -calculus interpreter, then this is usually written as the β -reduction $(\mathfrak{p} \mathfrak{d}) \rightarrow \mathfrak{r}$, where the left hand side is a λ -application, and \mathfrak{p} is defined by some λ -abstraction.

§4.2.12 DEFINITION For each program $\mathfrak{p} \in \mathfrak{P}$ we define a function $\mathfrak{F}_{\mathfrak{p}} : \mathfrak{E} \rightarrow \mathfrak{E}^+$, this way:

$$\mathfrak{F}_{\mathfrak{p}}(\mathfrak{e}) = \begin{cases} \mathfrak{r} & \text{if } \langle \mathfrak{p} \mathfrak{e} \rangle \hookrightarrow \mathfrak{r} \\ \infty & \text{if } \langle \mathfrak{p} \mathfrak{e} \rangle \hookrightarrow \infty . \end{cases}$$

COMMENT If $\forall \mathfrak{e} \in \mathfrak{E}, \mathfrak{F}_{\mathfrak{p}}(\mathfrak{e}) = \mathfrak{F}_{\mathfrak{q}}(\mathfrak{e})$, then we say that programs \mathfrak{p} and \mathfrak{q} are behaviorally equivalent, $\mathfrak{p} \equiv_{\mathfrak{F}} \mathfrak{q}$, or that \mathfrak{p} and \mathfrak{q} implement the same function.

§4.2.13 THEOREM $\forall \mathcal{P} \in \mathfrak{T}, F_{\mathcal{P}} = \mathfrak{F}_{\mathfrak{p}}$, where $\mathfrak{p} = c(\mathcal{P})$.

PROOF $\forall \mathfrak{d} \in \mathfrak{E}, \forall \mathcal{P} \in \mathfrak{T}, \mathfrak{F}_{\mathfrak{p}}(\mathfrak{d}) = F_{\mathcal{P}}(\mathfrak{d})$, see §4.1.8, because $\mathcal{U}\langle c(\mathcal{P}) \mathfrak{d} \rangle = \mathcal{P}(\mathfrak{d})$, by Theorem §4.2.1, and therefore $F_{\mathcal{P}} = \mathfrak{F}_{\mathfrak{p}}$ when the universal computer is a universal Turing machine, \mathcal{U} . Theorem §4.2.9 extends it to every universal computer. \diamond

COMMENT \mathcal{P} and \mathfrak{p} implement the same function.

COMMENT This theorem is a consequence of the program isomorphism, see §4.1.12. In other words, $\mathfrak{T} \cong \mathfrak{P}$ implies that $\equiv_F \leftrightarrow \equiv_{\mathfrak{F}}$, so $\mathcal{P} \equiv_F \mathcal{Q} \Leftrightarrow \mathfrak{p} \equiv_{\mathfrak{F}} \mathfrak{q}$.

COROLLARY $F_{\mathcal{U}} = \mathfrak{F}_{\mathfrak{u}}$, where $\mathfrak{u} = c(\mathcal{U})$.

§4.3 Turing's Thesis

§4.3.1 THESIS What is effectively calculable is computable.

COMMENT This is Church's thesis, or rather Turing's thesis, as it was expressed by Gandy (1980). There, 'something is effectively calculable' if its results can be found by some purely mechanical process, see §3.1.5, and 'computable' means that the same results will be found by some Turing machine. Then, ${}^*\mathbb{F} \subseteq \mathfrak{T}$.

COMMENT 'What is computable is effectively calculable', or $\mathfrak{T} \subseteq {}^*\mathbb{F}$, is the converse of Turing's thesis. And it is obvious that if a Turing machine can compute a function, then the function is effectively calculable, see §3.1.5, by a Turing machine. Therefore, ${}^*\mathbb{F} = \mathfrak{T}$, and $|{}^*\mathbb{F}| = \aleph_0$, by §4.1.13.

§4.3.2 REMARK An effectively calculable function is not an input to output mapping; it is a process to calculate the output from the input.

EXAMPLE To multiply a number expressed in binary by two we can append a '0' to it, which is an effectively calculable function that we will call *app0*. But the complete memoization of the same function, which we will call *memoby2*, is not effectively calculable because it would require an infinite quantity of memory. And therefore, $\text{app0} \neq \text{memoby2}$.

§4.3.3 NOTATION We will call every universe where the Turing's thesis is true a Turing universe. When we want to note that something is true in a Turing universe, we will use an asterisk, so $A \stackrel{*}{=} B$ means that $A = B$ if the Turing's thesis stands.

EXAMPLES ${}^*\mathbb{F} \stackrel{*}{=} \mathfrak{T}$ and $|{}^*\mathbb{F}| \stackrel{*}{=} \aleph_0$.

COMMENT The Turing's thesis affirms that this is a Turing universe. In any Turing universe the Turing's thesis is a law of nature, as it was defended by Post (1936), last paragraph. Then a Turing universe can also be called a Post universe.

COMMENT While the Turing's thesis is true, you can ignore the asterisks.

§4.3.4 THEOREM Universal computers are* the most capable computing devices.

PROOF If Turing's thesis stands, see §4.3.1, then anything that any mechanism can effectively calculate can be computed by some Turing machine, and therefore, by Theorem §4.2.1, it can be computed by any universal Turing machine, and finally, by Theorem §4.2.9, it can be computed by any universal computer. \diamond

§4.3.5 LEMMA There are definable functions that no Turing machine can compute.

PROOF You can use a diagonal argument, or work from other theorems that use the diagonal argument. For example, the set of Turing machines is countable, see §4.1.13, $|\mathfrak{T}| = |\mathbb{N}| = \aleph_0$, while the possible number of predicates on natural numbers, that is, the number of functions $\mathbb{N} \rightarrow \mathbb{B}$, is $2^{|\mathbb{N}|} = 2^{\aleph_0}$, which is not countable, $|\mathfrak{T}| = |\mathbb{N}| = \aleph_0 < 2^{\aleph_0} = 2^{|\mathbb{N}|}$. This uses Cantor's theorem, $|S| < |2^S|$, with its diagonal argument. So there are not enough Turing machines to compute every definable function. \diamond

COROLLARY Universal computers cannot compute every definable function.

COMMENT If the Turing's thesis stands, see §4.3.1, then it follows that there are definable functions that are not effectively calculable, see §3.1.5.

COMMENT There are* more mappings than processes.

§4.3.6 DEFINITION The identity Turing machine, \mathcal{I} , just halts.

COMMENT It does nearly nothing. But, wait!

§4.3.7 LEMMA $\forall \mathfrak{x} \in \mathfrak{E}, \mathcal{I}\langle \mathfrak{x} \rangle \hookrightarrow \mathfrak{x}$, where \mathcal{I} is the identity Turing machine.

PROOF Whatever expression $\mathfrak{x} \in \mathfrak{E}$ is written on the tape of \mathcal{I} , that very same expression \mathfrak{x} is written when \mathcal{I} halts, because halting is all what \mathcal{I} does. \diamond

COMMENT \mathcal{I} does not touch the expression.

§4.3.8 LEMMA The identity Turing machine is terminating, that is, $\mathcal{I} \in \Sigma_\sigma$.

PROOF The identity Turing machine, which just halts, is terminating, see §4.2.5, because it always halts; it only halts. \diamond

COMMENT \mathcal{I} behaves, because sometimes ‘you can look, but you better not touch’.

§4.3.9 LEMMA The identity Turing machine $\mathcal{I} : \mathfrak{E} \rightarrow \mathfrak{E}$ is* the identity function $i : \mathbb{S} \rightarrow \mathbb{S}$ such that $\forall x \in \mathbb{S}, i(x) = x$, that is, $\mathcal{I} \stackrel{*}{=} i$.

PROOF The identity function i is an effectively calculable function, see §3.1.5. Therefore, if the Turing’s thesis stands, see §4.3.1, then there is a Turing machine \mathcal{J} such that $\forall \mathfrak{x} \in \mathfrak{E}, \mathcal{J}\langle \mathfrak{x} \rangle \hookrightarrow \mathfrak{x}$. By Lemma §4.3.7, that Turing machine \mathcal{J} is the identity Turing machine \mathcal{I} . \diamond

COMMENT If $\mathfrak{i} = \mathfrak{c}(\mathcal{I})$, then $\mathcal{U}\langle \mathfrak{i} \mathfrak{p} \mathfrak{d} \rangle = \mathcal{I}\langle \mathfrak{p} \mathfrak{d} \rangle \hookrightarrow \mathfrak{p} \mathfrak{d}$, and $\mathcal{U}\langle \mathfrak{u} \mathfrak{p} \mathfrak{d} \rangle = \mathcal{U}\langle \mathfrak{p} \mathfrak{d} \rangle = \mathcal{P}\langle \mathfrak{d} \rangle \hookrightarrow \mathfrak{r}$, or ∞ , see §4.2.3. Then \mathfrak{i} is the literal identity for expressions, or quotation, and \mathfrak{u} is the functional identity for programs, or evaluation. Both are computable, but $\mathcal{I} \in \Sigma_\sigma$ and $\mathcal{U} \in \Sigma_{\bar{\sigma}}$, see §4.2.7.

§4.3.10 THEOREM Everything is* an expression, that is, $\mathfrak{E} \stackrel{*}{=} \mathbb{S}$.

PROOF \mathbb{S} is the set of everything, see §3.2.1. In computing, there are only computing devices, \mathfrak{T} , and expressions, \mathfrak{E} , see §4.1.4. But then, by the program isomorphism, see §4.1.12, computing devices are expressions, $\mathfrak{T} \subset \mathfrak{E}$. Therefore, in computing everything is an expression. And now, if the Turing’s thesis stands, see §4.3.1, then Lemma §4.3.9 also stands, so $\forall x \in \mathbb{S}, x = i(x) \stackrel{*}{=} \mathcal{I}\langle x \rangle \hookrightarrow x \in \mathfrak{E}$. The converse, $\forall \mathfrak{x} \in \mathfrak{E}, \mathfrak{x} \leftarrow \mathcal{I}\langle \mathfrak{x} \rangle = i(\mathfrak{x}) = \mathfrak{x} \in \mathbb{S}$, holds irrespective of Turing’s thesis. Therefore, $\mathbb{S} \stackrel{*}{=} \mathfrak{E}$. \diamond

COMMENT We will write \mathfrak{x} to indicate a computing point of view of x , but $\forall x, x \stackrel{*}{=} \mathfrak{x}$. For example, $i \stackrel{*}{=} \mathfrak{i}$.

§4.3.11 LEMMA The set of solutions \mathbb{S} is* countable, that is, $|\mathbb{S}| \stackrel{*}{=} \aleph_0$.

PROOF $\mathbb{S} \stackrel{*}{=} \mathfrak{E}$, by §4.3.10, and $|\mathfrak{E}| = |\mathbb{N}| = \aleph_0$, by §4.1.5, therefore $|\mathbb{S}| \stackrel{*}{=} |\mathfrak{E}| = \aleph_0$. \diamond

COMMENT We will refer to the set of solutions in a Turing universe as \mathbb{S}^* . So we can also write this lemma as $|\mathbb{S}^*| = \aleph_0$.

§4.3.12 THEOREM Resolving is* computing, that is, $\mathfrak{T} \stackrel{*}{=} \mathbb{R}$.

PROOF From Theorem §4.3.10, everything is* an expression, and taking transitions and not states, it follows that whatever transforms expressions in computing theory, that is, a Turing machine \mathcal{P} , or its equivalent program \mathfrak{p} , or a λ -function of the λ -calculus, is* equivalent to whatever transforms sets in set theory, that is, an effectively calculable function, and it is* also equivalent to whatever transforms problems in problem theory, that is, a resolution \mathfrak{R} . Therefore, resolving is* computing, $\mathbb{R} \stackrel{*}{=} \mathfrak{T}$. \diamond

COMMENT $*f \equiv \mathcal{P} \equiv \mathfrak{p} \equiv \mathfrak{R}$, and $*\mathbb{F} \equiv \mathfrak{T} \equiv \mathbb{R}$.

COMMENT We can define functions that are not effectively calculable, see §4.3.5. Those functions that cannot effectively calculate, cannot therefore transform, and they are, in this sense, useless; we can define them, but we cannot use them.

COROLLARY Metasolutions are* effectively calculable functions, that is, $\Pi\mathbb{S} \stackrel{*}{=} \mathbb{F}$.

PROOF Because $\mathbb{R} = \Pi\mathbb{S}$, see §3.5.8, so $\Pi\mathbb{S} = \mathbb{R} \stackrel{*}{=} \mathbb{F}$. \diamond

§4.3.13 **LEMMA** The set of resolutions \mathbb{R} is* countable, that is, $|\mathbb{R}| \stackrel{*}{=} \aleph_0$.

PROOF $\mathbb{R} \stackrel{*}{=} \mathfrak{T}$, by §4.3.12, and $|\mathfrak{T}| = |\mathbb{N}| = \aleph_0$, by §4.1.13, therefore $|\mathbb{R}| \stackrel{*}{=} |\mathfrak{T}| = \aleph_0$. \diamond

COMMENT We will refer to the set of resolutions in a Turing universe as \mathbb{R}^* . So we can also write this lemma as $|\mathbb{R}^*| = \aleph_0$.

§4.3.14 **LEMMA** Predicate P_{δ_s} , where $P_{\delta_s}(x) = [x = s]$, is* effectively calculable.

PROOF Both s and x are* expressions, by §4.3.10, so both are finite strings of symbols, $s = s_1s_2 \dots s_n$, and $x = x_1x_2 \dots x_m$. Then we can define a Turing machine with $n+2$ states, that starts in state 1, and that when some string x is written on its tape, it scans the string x , symbol by symbol, from the leftest one, this way: 1) in state i , with $1 \leq i \leq n$, if the read symbol is s_i , then it writes a *blank*, goes to state $i+1$, and moves to the *right*, but if the read symbol is not s_i , then it writes a *blank*, goes to state 0, and moves to the *right*; 2) in state $n+1$, if the read symbol is *blank*, then it writes a \top , goes to state 0, and *halts*, but if the read symbol is not *blank*, then it writes a *blank*, goes to state 0, and moves to the *right*; 3) in state 0, if the read symbol is not *blank*, then it writes a *blank*, goes to state 0, and moves to the *right*, but if the read symbol is *blank*, then it writes a \perp , goes to state 0, and *halts*. This Turing machine implements P_{δ_s} , and therefore P_{δ_s} is computable. \diamond

COROLLARY Problem $\delta_s = x? [x = s]$ is* expressible.

PROOF Because problem δ_s condition P_{δ_s} is* effectively calculable, see §3.1.6. \diamond

COMMENT Problem δ_s is used in the proof of Theorem §3.2.1.

COROLLARY The only solution to problem δ_s is s , so $\Sigma_{\delta_s} = \{s\} \in \mathbb{S}^1$.

PROOF Because $\Sigma_{\delta_s} = \{x \mid P_{\delta_s}(x)\} = \{x \mid [x = s]\} = \{s\}$. \diamond

§4.3.15 **LEMMA** The set of problems \mathbb{P} is* countable, that is, $|\mathbb{P}| \stackrel{*}{=} \aleph_0$.

COMMENT We will refer to the set of problems in a Turing universe as \mathbb{P}^* . So we can also write this lemma as $|\mathbb{P}^*| = \aleph_0$. If the condition of a problem is computable, then the problem is in \mathbb{P}^* ; $\delta_s \in \mathbb{P}^*$, for example.

PROOF Problem δ_s is* expressible, see §4.3.14. Then, $\delta_{\mathbb{S}^*} = \{\delta_s \mid s \in \mathbb{S}^*\} \subseteq \mathbb{P}^*$ because each $\delta_s \in \mathbb{P}^*$, and $|\delta_{\mathbb{S}^*}| = |\mathbb{S}^*|$ because there is a bijection $\delta_{\mathbb{S}^*} \leftrightarrow \mathbb{S}^* : \delta_s \leftrightarrow s$. Also, by Theorem §4.3.10, $\mathbb{P}^* \subseteq \mathfrak{E}$. Therefore, $\delta_{\mathbb{S}^*} \subseteq \mathbb{P}^* \subseteq \mathfrak{E}$, and $|\delta_{\mathbb{S}^*}| = |\mathbb{S}^*| = \aleph_0 = |\mathfrak{E}|$, and then, by the Cantor-Bernstein-Schröder theorem, $|\mathbb{P}^*| = \aleph_0$. \diamond

COMMENT The Cantor-Bernstein-Schröder theorem is Theorem B of §2, page 484, in Cantor (1895). We have really used the equivalent Theorem C, in the same page.

§4.3.16 **THEOREM** All problem sets are* countable, that is, $|\mathbb{S}^*| = |\mathbb{P}^*| = |\mathbb{R}^*| = \aleph_0$.

PROOF By Lemmas §4.3.11, §4.3.15, and §4.3.13. \diamond

§4.4 Full Resolution Machine

§4.4.1 **DEFINITION** A full resolution machine is a device that can execute any resolution.

§4.4.2 **THEOREM** A full resolution machine is* a Turing complete device.

PROOF By Theorem §4.3.12, resolving is* computing, $\mathfrak{R} \stackrel{*}{=} \mathcal{P}$. This means that to achieve the maximum resolving power is* to achieve the maximum computing power, which is* the computing power of a universal computer, by Theorem §4.3.4. Therefore, in a Turing universe a full resolution machine has to be Turing complete. \diamond

COMMENT To execute any resolution $\mathfrak{R} : \mathbb{P} \rightarrow 2^{\mathbb{S}}$, the full resolution machine has to calculate functions that can take functions and that can return functions without limitations, as $2^{2^{\mathbb{S}} \rightarrow 2^{\mathbb{S}}} \rightarrow 2^{2^{\mathbb{S}} \rightarrow 2^{\mathbb{S}}}$ for meta-analogies, see §3.6.9. Then a full resolution machine has to execute every possible function, and therefore, in a Turing universe, it has to execute every computable function, and then it has to be a λ -calculus interpreter, or an equivalent computing device, for example \mathcal{U} .

COMMENT This means that problem resolving is* equal to computing, and then full problem resolving is* equal to universal computing.

COROLLARY A full resolution machine* is a universal computer.

PROOF Because a full resolution machine is* a universal computer. \diamond

COMMENT Now we will state two equivalences between computing theory and problem theory concepts that are true in any Turing universe, and that are needed to show the limitations of full resolution machines.

§4.4.3 DEFINITION A set is recursively enumerable if there is a Turing machine that generates all of its members, and then halts.

COMMENT If the set is infinite, the Turing machine will keep generating its members forever.

DEFINITION A set is computable if it is recursively enumerable.

§4.4.4 THEOREM Resolvable in problem theory is* equivalent to recursively enumerable in computing theory, that is,

$$\text{Resolvable} \stackrel{*}{=} \text{Recursively Enumerable} .$$

PROOF To see that a problem is resolvable if, and only if, the set of its solutions is recursively enumerable, just compare the definition of resolvable problem, in §3.5.4, with the definition of recursively enumerable set, in §4.4.3. The only remaining gap is to equate the valid resolution \mathfrak{R} of the resolvable problem to the Turing machine of the recursively enumerable set, a gap that we can bridge with the help of Theorem §4.3.12. Finally see that, by the set isomorphism, see §3.2.9, we can refer interchangeably to the problem π or to the set of its solutions Σ_{π} . Then we can say that a problem is recursively enumerable, or that a set is resolvable. \diamond

§4.4.5 DEFINITION A set is recursive if its characteristic function can be computed by a Turing machine that always halts.

§4.4.6 THEOREM Expressible in problem theory is* equivalent to recursive in computing theory, that is,

$$\text{Expressible} \stackrel{*}{=} \text{Recursive} .$$

PROOF The condition of a problem, P_{π} , is the characteristic function of the set of its solutions, because $\Sigma_{\pi} = \{s \mid P_{\pi}(s)\}$, see §3.2.2. Then, if the set of all the solutions to a problem is a recursive set, see §4.4.5, then the condition P_{π} can be computed by a Turing machine that always halts. So the condition P_{π} is an effectively calculable function, and therefore the problem is expressible, see §3.1.6. If Turing's thesis, §4.3.1, is true, then the converse is also true; just go backwards from expressible to recursive. Finally, by the set isomorphism, see §3.2.9, we can refer interchangeably to the problem π or to the set of its solutions Σ_{π} . Then, we can say that a problem is recursive, or that a set is expressible. \diamond

§4.4.7 LEMMA The limitations of full resolution machines are* the limitations of universal computers.

PROOF Because a full resolution machine is* a universal computer, see §4.4.2. \diamond

COMMENT Even if universal computers are the most capable computers, they cannot compute everything, see §4.3.5. Now we will present three limits related to problems.

§4.4.8 LEMMA A full resolution machine can* execute any resolution, but it cannot* express some problems.

PROOF There is a recursively enumerable set that is not recursive; this is the last theorem in Post (1944) §1. Translating, by Theorems §4.4.4 and §4.4.6, to problem theory: there is* a resolvable problem that is* not expressible. \diamond

COMMENT This is the problem limit of full resolution machines*.

COMMENT That last theorem in Post (1944) §1, page 291, is an abstract form of Gödel's incompleteness theorem, see Post (1944) §2.

§4.4.9 LEMMA A full resolution machine can* execute any resolution, but it cannot* resolve some problems.

PROOF Let us call κ some problem that is resolvable but not expressible, see §4.4.8. This means that $\exists \mathfrak{R} \mid \mathfrak{R}(\kappa) = \Sigma_\kappa$, but $\nexists P_\kappa \mid P_\kappa(x) = [x \in \Sigma_\kappa]$. Note that $|\Sigma_\kappa| \geq \aleph_0$, because otherwise $\exists P_\kappa$. Then its metaproblem $\Pi\kappa$ is solvable but not resolvable. $\Pi\kappa$ is solvable because κ is resolvable, see §3.5.12, or, easier, because \mathfrak{R} is a solution to $\Pi\kappa$. For $\Pi\kappa$ to be resolvable there should be a resolution that would find 'all the solutions of $\Pi\kappa$ ', that is, 'all the valid resolutions for κ '. But, whenever a possible valid resolution for κ , let us call it \mathfrak{R}' , generates a value not yet generated by \mathfrak{R} , let us call it z , we cannot decide whether $z \in \Sigma_\kappa$ and it will be eventually generated by \mathfrak{R} , or if $z \notin \Sigma_\kappa$ and it will never be generated by \mathfrak{R} ; remember that κ is not expressible, $\nexists P_\kappa$. And, not being able to decide on z , we cannot decide whether \mathfrak{R}' is a valid resolution for κ or not. \diamond

COMMENT This is the resolution limit of full resolution machines*.

COMMENT Problem κ is named after the complete set K of Post (1944), §3.

§4.4.10 LEMMA A full resolution machine can execute any resolution, but it cannot solve some problems.

PROOF Simply because some problems have not any solution, $\Sigma_\pi = \{\} = \emptyset$. \diamond

COMMENT This is the solution limit of full resolution machines, which also applies to full resolution machines*.

COMMENT An unsolvable problem can be resolved by showing that it has not any solution. For example, the decision problem of the halting problem, $\Delta\eta$, see §4.5.4 below, was resolved unsolvable by Turing (1936), §8.

§4.4.11 THEOREM A full resolution machine* can execute any resolution, but it cannot express some problems (problem limit), and it cannot resolve some problems (resolution limit), and it cannot solve some problems (solution limit).

PROOF By Lemmas §4.4.8, §4.4.9, and §4.4.10. \diamond

COMMENT Full resolution machines* have limitations on each of the three main concepts of the problem theory.

$$\text{Problem} \xrightarrow{\text{Resolution}} \{ \text{Solution} \}$$

§4.5 Problem Topology

§4.5.1 DEFINITION The decision problem of a problem $\pi = x? P_\pi(x)$, written $\Delta\pi$, is:

$$\Delta\pi = \mathcal{P}? [\mathcal{P} \in \Sigma_\sigma] \wedge [\forall(\mathfrak{x} \stackrel{*}{=} x), \mathcal{P}(\mathfrak{x}) \stackrel{*}{=} P_\pi(x)].$$

COMMENT A solution to the decision problem $\Delta\pi$ of some original problem π is a Turing machine \mathcal{P} that always halts and that computes the original problem condition P_π for any input. Decision problems are only defined in Turing universes, where $\mathfrak{x} \stackrel{*}{=} x$ by Theorem §4.3.10.

COMMENT This definition follows Post (1944), page 287.

§4.5.2 DEFINITION The halting condition $P_\eta : \mathfrak{T} \times \mathfrak{E} \rightarrow \mathbb{B}$ is:

$$P_\eta(\mathcal{P}, \mathfrak{d}) = \begin{cases} \perp & \text{if } \mathcal{P}(\mathfrak{d}) \hookrightarrow \infty \\ \top & \text{otherwise .} \end{cases}$$

§4.5.3 DEFINITION The halting problem is $\eta = (p, d)? P_\eta(p, d)$.

COMMENT The halting problem η corresponds to the halting condition P_η by the condition isomorphism of problems, see §3.1.7.

COMMENT $P_\sigma(p) = \bigwedge_{d \in \mathfrak{E}} P_\eta(p, d)$, see §4.2.5, so $\sigma = \bigwedge_{d \in \mathfrak{E}} \eta$, by §3.1.7 and §3.1.10.

§4.5.4 DEFINITION The decision problem of the halting problem, $\Delta\eta$, is:

$$\Delta\eta = \mathcal{H}? [\mathcal{H} \in \Sigma_\sigma] \wedge [\forall \mathcal{P} \in \mathfrak{T}, \forall \mathfrak{d} \in \mathfrak{E}, \mathcal{H}(\mathfrak{c}(\mathcal{P}) \mathfrak{d}) = P_\eta(\mathcal{P}, \mathfrak{d})].$$

§4.5.5 THEOREM The decision problem of the halting problem $\Delta\eta$ has not any solution.

PROOF Turing (1936), §8, resolved that $\Delta\eta$ is unsolvable. \diamond

COMMENT There is not any Turing machine that always halts and that compute P_η for each possible input. There is not any algorithm $\mathfrak{a} \in \mathfrak{A}$ that would compute $P_\eta(\mathfrak{p}, \mathfrak{d})$ for every pair $(\mathfrak{p}, \mathfrak{d}) \in \mathfrak{P} \times \mathfrak{E}$.

§4.5.6 LEMMA The decision problem $\Delta\pi$ of some problem π is solvable if, and only if, the problem π is expressible*, that is, $\Delta\pi$ is solvable $\Leftrightarrow \pi$ is expressible*.

PROOF From solvable to expressible. That the decision problem $\Delta\pi$ is solvable, see §4.5.1, means that there is a Turing machine that always halts, and that computes P_π for each possible input. Therefore, P_π is effectively calculable, see §3.1.5, by a Turing machine, and then the problem π is expressible, see §3.1.6, and then it is also expressible*. Now from expressible to solvable. If a problem π is expressible, then its condition P_π is an effectively calculable function, see §3.1.6. Then, if the Turing's thesis stands, see §4.3.1, that is, if it is expressible*, then there is a Turing machine \mathcal{P} that can compute P_π exactly as the effectively calculable function. \mathcal{P} always halts, because P_π is a condition, so its result is finite. Therefore, the decision problem $\Delta\pi$ of the problem has a solution, \mathcal{P} , and then $\Delta\pi$ is solvable, see §3.2.14. \diamond

COROLLARY The halting problem η is not expressible*.

PROOF The decision problem of the halting problem, $\Delta\eta$, is not solvable, see §4.5.5, and then the halting problem η is not expressible*. \diamond

COMMENT The halting problem η is inexpressible*, but solvable. While the decision problem of the halting problem $\Delta\eta$ is unsolvable, the halting problem η has many solutions.

§4.5.7 THEOREM The following equivalences stand:

$$\begin{aligned}\Delta\pi \text{ is solvable} &\Leftrightarrow \pi \text{ is expressible,} \\ \Pi\pi \text{ is solvable} &\Leftrightarrow \pi \text{ is resolvable,} \\ \pi \text{ is solvable} &\Leftrightarrow \pi \text{ is solvable.}\end{aligned}$$

PROOF The last one is trivial, and the other two equivalences were already proved by Lemmas §4.5.6 and §3.5.12. \diamond

§4.5.8 DEFINITION A problem π can be: expressible* (\mathcal{E}) or not expressible* ($\bar{\mathcal{E}}$), resolvable* (\mathcal{R}) or not resolvable* ($\bar{\mathcal{R}}$), and solvable (\mathcal{S}) or not solvable ($\bar{\mathcal{S}}$).

COMMENT An expressible problem is* equivalent to a recursive set, by Theorem §4.4.6, a resolvable problem is* equivalent to a recursively enumerable set, by Theorem §4.4.4, and an unsolvable problem is equivalent to an empty set.

COMMENT Then \mathcal{R} is the set of computable sets, see §4.4.3.

COMMENT Not every combination is possible.

§4.5.9 LEMMA If a problem is expressible*, then it is resolvable*, that is, $\mathcal{E} \subset \mathcal{R}$.

PROOF Because every recursive set is recursively enumerable, $\mathcal{E} \subseteq \mathcal{R}$. This is a corollary to the first theorem in Post (1944) §1. And $\mathcal{E} \neq \mathcal{R}$, see the proof of Lemma §4.4.8. To translate between sets and problems we use Theorems §4.4.4 and §4.4.6. \diamond

COMMENT The first theorem in Post (1944) §1, page 290, states that a set M is recursive if and only if both the set M and its complement \bar{M} are recursively enumerable.

§4.5.10 LEMMA If a problem is not solvable, then it is expressible*, that is, $\bar{\mathcal{S}} \subset \mathcal{E}$.

PROOF If a problem ν is not solvable, $\nu \in \bar{\mathcal{S}}$, then $\Sigma_\nu = \{\}$, see §3.2.14. So ν is a contradictory problem, see §3.1.11, and its condition P_ν is the contradiction $P_{\bar{\tau}}$, that is, $\forall x, P_\nu(x) = P_{\bar{\tau}}(x) = \perp$. So $P_\nu = P_{\bar{\tau}}$ is an effectively calculable function, see §3.1.5, and therefore ν is expressible, see §3.1.6, and then expressible*. And $\bar{\mathcal{S}} \neq \mathcal{E}$, because $(x? [2x = x^2]) \in \mathcal{S} \cap \mathcal{E}$. \diamond

COMMENT Being expressible*, by Lemma §4.5.9, ν is also resolvable*: $\bar{\mathcal{S}} \subset \mathcal{E} \subset \mathcal{R}$.

§4.5.11 THEOREM Regarding expressibility* \mathcal{E} , resolvability* \mathcal{R} , and solvability \mathcal{S} , the topology of the problem space is:

$$\bar{\mathcal{S}} \subset \mathcal{E} \subset \mathcal{R} \subset \mathbb{P}.$$

PROOF By Lemmas §4.5.9 and §4.5.10. As shown in the table, these lemmas prevent four of the eight combinations, and the examples show that the other four do exist.

| \mathcal{E} | \mathcal{R} | \mathcal{S} | Example & Comment |
|---------------|---------------|---------------|-----------------------------------|
| \top | \top | \top | $x? [2x = x^2]$ |
| \top | \top | \perp | $x? [2x = x^2] \wedge [x > 2]$ |
| \top | \perp | \top | None, by Lemma §4.5.9 |
| \top | \perp | \perp | None, by Lemma §4.5.9 |
| \perp | \top | \top | κ , see §4.4.9 |
| \perp | \top | \perp | None, by Lemma §4.5.10 |
| \perp | \perp | \top | $\Pi\kappa$, see §4.4.9 |
| \perp | \perp | \perp | None, by Lemma §4.5.10 \diamond |

COROLLARY Then, $\{\bar{\mathcal{S}}, \mathcal{E} \cap \mathcal{S}, \mathcal{R} \cap \bar{\mathcal{E}}, \bar{\mathcal{R}}\}$ is a partition of \mathbb{P} .

COMMENT See below, in §6.2, that $\mathcal{E} = \mathbb{P}$, and then $\mathbb{P}^* \subset \mathbb{P}$. Also $\mathcal{R} = 2^{\mathbb{S}}$.

§4.5.12 DEFINITION We say that a problem is finite, if the set of its solutions is finite. We will refer to the set of finite problems as \mathcal{F} . That is, $\mathcal{F} = \{ \pi \mid |\Sigma_\pi| < \aleph_0 \}$.

§4.5.13 LEMMA The set of finite problems \mathcal{F} is a proper subset of the set of expressible problems \mathcal{E} . The set of not solvable problems $\overline{\mathcal{S}}$ is a proper subset of the set of finite problems \mathcal{F} . That is, $\overline{\mathcal{S}} \subset \mathcal{F} \subset \mathcal{E}$.

PROOF $\mathcal{F} \subset \mathcal{E}$ because all finite sets are recursive, but not the converse. $\overline{\mathcal{S}} \subset \mathcal{F}$ because $\forall \nu \in \overline{\mathcal{S}}, |\Sigma_\nu| = 0 < \aleph_0$, but $(x? [2x = x^2]) \in \mathcal{S} \cap \mathcal{F}$. \diamond

PROPOSITION Including \mathcal{F} , the topology of \mathbb{P} is: $\overline{\mathcal{S}} \subset \mathcal{F} \subset \mathcal{E} \subset \mathcal{R} \subset \mathbb{P}$.

COROLLARY The topology $\overline{\mathcal{S}} \subset \mathcal{F} \subset \mathcal{E} \subset \mathcal{R} \subset \mathbb{P}$ partitions the problem space \mathbb{P} into five non-empty places: $\overline{\mathcal{S}}, \mathcal{F} \cap \mathcal{S}, \mathcal{E} \cap \overline{\mathcal{F}}, \mathcal{R} \cap \overline{\mathcal{E}}$, and $\overline{\mathcal{R}}$.

§4.5.14 REMARK The upper part of this topology is further refined by the so called Turing degrees of unsolvability, that we will call Turing degrees of inexpressibility. Turing degree zero, $\mathbf{0}$, corresponds to the first three places, because $\mathcal{E} = \mathbf{0}$.

COMMENT Then, $|\mathcal{E}| = |\mathbf{0}| = \aleph_0$, $|\mathcal{R}| = \aleph_0$, and $|\mathbb{P}| = 2^{\aleph_0} > \aleph_0$. To complete the cardinalities, $|\overline{\mathcal{S}}| = 1$, so $|\mathcal{S}| = 2^{\aleph_0}$, and $|\mathcal{F}| = \aleph_0$.

§4.5.15 REMARK Noting \mathcal{E}^p the set of problems defined by a condition that can be computed in polynomial time, and \mathcal{R}^p the set of problems that can be resolved in polynomial time, then $\mathcal{E}^p \subset \mathcal{E}$ and $\mathcal{R}^p \subset \mathcal{R}$. The so called ‘P = NP?’ question asks if $\mathcal{E}^p = \mathcal{R}^p$, because $\mathbf{P} = \mathcal{E}^p$ and $\mathbf{NP} = \mathcal{R}^p$, and then it should be called the ‘ $\mathcal{E}^p = \mathcal{R}^p$?’ question. See that the general question ‘ $\mathcal{E} = \mathcal{R}$?’ was answered negatively by Lemma §4.5.9, because $\mathcal{E} \subset \mathcal{R}$, and that $\mathcal{E}^p \subseteq \mathcal{R}^p$.

COMMENT A similar question is ‘ $\mathcal{E}^p \setminus \{\emptyset\} = \mathcal{S}^p$?’ where \mathcal{S}^p is the set of problems that can be solved in polynomial time, so $\mathcal{S}^p \subset \mathcal{S}$. The corresponding general question is also answered negatively, because $\mathbb{P} = \mathcal{S} \cup \{\emptyset\}$, so $\mathcal{E} \setminus \{\emptyset\} \subset \mathcal{S}$.

§5 Resolvers

§5.1 Semantics and Syntax

§5.1.1 REMARK In this Section §5, we will always be inside a Turing universe, see §4.3.3, and accordingly we will drop every asterisk. Though some results do not depend on Turing’s thesis, by now the reader should know when it is the case.

§5.1.2 DEFINITION A resolver is a device that takes problems and returns solutions.

COMMENT A resolver executes resolutions.

COMMENT After Theorem §4.3.12, we can equate a resolution $\mathfrak{R} \in \mathbb{R}$ to the computing device that executes the resolution $\mathcal{P} \in \mathfrak{T}$, that is, $\mathfrak{R} = \mathcal{P}$.

§5.1.3 DEFINITION We will call the domain of \mathbb{S} semantics. We will call the domain of $\mathbb{S} \rightarrow \mathbb{S}$ syntax.

COMMENT As λ -calculus shows, we only need functions to implement any syntax.

COMMENT By Theorem §3.2.1, everything is in \mathbb{S} , including $\mathbb{S} \rightarrow \mathbb{S}$. But this is both mathematically impossible, by Cantor’s theorem, and practically not interesting.

EXAMPLE Using a practical example, if the problem is the survival problem, so some behaviors keep the resolver alive, and the rest cause the death of the resolver, then \mathbb{S}

is the set of behaviors, and it does not include anything that is not a behavior, not even predicates on behaviors, nor functions. Note that the condition of the survival problem, which is satisfied if the resolver does not die, is a predicate on behaviors.

§5.1.4 REMARK In this Section §5, we will assume that \mathbb{S} is not the set of everything, and, in particular, we will assume that there is not any function in \mathbb{S} . We will focus on the survival problem, and then assume that \mathbb{S} is the set of behaviors, or finite state automata, but you can think that $\mathbb{S} = \mathbb{N}$, or any other countable set, see §4.3.11. Then we will build a series of resolvers, from the simplest one implementing one element of \mathbb{S} , to more complex resolvers that have to implement functions in order to look for resolutions to deal with metaproblems.

§5.1.5 DEFINITION A problem type, for example \mathbb{P}_Ψ , is a subset of the set of problems, that is, $\mathbb{P}_\Psi \subseteq \mathbb{P}$. We will note \mathbb{S}_Ψ the set of the solutions to the type of problems \mathbb{P}_Ψ . That is, $\forall \pi_\Psi \in \mathbb{P}_\Psi, \Sigma_{\pi_\Psi} \subseteq \mathbb{S}_\Psi \subseteq \mathbb{S}$.

COMMENT The survival problem is not a single problem, but a type of problems, \mathbb{P}_Ω ; each living being faces a different survival problem. But, in this case as in many others, what it is certain is that the solutions to any of these problems is of a specific kind. For example, while eating can be a solution, imagining how to eat is not a solution, even though it can help us to get something to eat, because it can be a metasolution. Then \mathbb{S}_Ω is the set of behaviors.

§5.1.6 REMARK Metaproblems $\Pi\pi$ are a type of problem, $\Pi\mathbb{P} = \mathbb{P}_\Pi$, and its solutions are resolutions, $\Pi\mathbb{S} = \mathbb{S}_\Pi = \mathbb{R}$, see §3.5.8.

§5.1.7 LEMMA If the set of the solutions to some type of problem is finite, $0 < |\mathbb{S}_\Gamma| < \aleph_0$, then each and every problem of that type is expressible and resolvable.

PROOF Because those problems are in \mathcal{F} , so Lemma §4.5.13 apply. \diamond

COMMENT If $0 < |\mathbb{S}_\Gamma| = N < \aleph_0$, then $|\mathbb{P}_\Gamma| = 2^N < \aleph_0$ and $|\mathbb{R}_\Gamma| = (2^N)^{2^N} < \aleph_0$. In the finite case, $|\mathbb{S}_\Gamma| < |\mathbb{P}_\Gamma| < |\mathbb{R}_\Gamma| < \aleph_0$.

§5.1.8 DEFINITION A constant function $K_s : \mathbb{S} \rightarrow \mathbb{S}$ is: $\forall s \in \mathbb{S}, \forall x \in \mathbb{S}, K_s(x) = s$.

COMMENT Every constant function K_s is effectively calculable, see §3.1.5. They are λ -definable; in λ -calculus, $K = (\lambda s x. s)$. This is because our λ -calculus includes the K combinator, and so we refer to the λK -calculus simply as λ -calculus.

SPECIAL CASES Tautology: $K_\top = P_\top$. Contradiction: $K_\perp = P_\perp$. See §3.1.11.

§5.1.9 DEFINITION The constant isomorphism is the natural isomorphism between \mathbb{S} and the set of constant functions \mathbb{K} that relates each $s \in \mathbb{S}$ with $K_s \in (\mathbb{S} \rightarrow \mathbb{S})$. That is, $\mathbb{S} \Leftrightarrow \mathbb{K} : s \leftrightarrow K_s$.

COMMENT We can extend any operation on \mathbb{S} to \mathbb{K} . For example, for any binary operation $*$ on \mathbb{S} , we define $\forall x, [K_a * K_b](x) = K_a(x) * K_b(x) = a * b = K_{a*b}(x)$.

COMMENT Semantics is included in syntax, that is, $\mathbb{S} \cong \mathbb{K} \subset (\mathbb{S} \rightarrow \mathbb{S})$.

§5.1.10 REMARK A semantic function $f : \mathbb{S} \rightarrow \mathbb{S}$ is a syntactic element, $f \in (\mathbb{S} \rightarrow \mathbb{S})$, but it is not a syntactic function $\mathfrak{f} \in ((\mathbb{S} \rightarrow \mathbb{S}) \rightarrow (\mathbb{S} \rightarrow \mathbb{S}))$, because the semantic function f takes semantic elements and returns semantic elements, while, using the constant isomorphism, the syntactic function \mathfrak{f} is not restricted. In particular, a semantic function cannot take a function, and a semantic function cannot return a function.

COMMENT In semantics, literal identity i is the identity, see §4.3.9, because there are not higher order functions in semantics. But, different syntactic objects can refer to the same semantic object, as in $f(x) = y$, which means that $f(x)$ and y are two syntactic objects that refer to the same semantic object. Then, there are two identities in syntax: literal identity i , or quotation, which is the semantic function that just returns what it takes, and functional identity u , or evaluation, which is the syntactic function that follows the references and returns the final one, see §4.2.3. Note also that a syntactic object can refer to no semantic object, and then we say that the syntactic object is a paradox.

§5.1.11 DEFINITION The range of a resolver \mathfrak{R} , noted $\Xi\mathfrak{R}$, is the set of the problems for which \mathfrak{R} provides a non-empty subset of solutions, and only of solutions, that is, $\Xi\mathfrak{R} = \{ \pi \mid \mathfrak{R}(\pi) \subseteq \Sigma_\pi \wedge \mathfrak{R}(\pi) \neq \emptyset \}$.

COMMENT The range of a resolver is the set of the problems that the resolver solves.

§5.1.12 DEFINITION The power of a resolver \mathfrak{R} , noted $\Phi\mathfrak{R}$, is the set of the problems that the resolver \mathfrak{R} resolves, that is, $\Phi\mathfrak{R} = \{ \pi \mid \mathfrak{R}(\pi) = \Sigma_\pi \}$.

COMMENT In practice, if $|\Sigma_\pi| > 1$, it is not sensible to generate all the solutions, Σ_π , when just one solution solves the problem. In these cases the range of the resolver is more important than its power.

§5.1.13 THEOREM $\forall \mathfrak{R}, \mathcal{S} \cap \Phi\mathfrak{R} \subseteq \Xi\mathfrak{R}$.

PROOF Because $\forall \pi \in \Phi\mathfrak{R} \cap \mathcal{S}$, we have that $\pi \in \Phi\mathfrak{R}$ so $\mathfrak{R}(\pi) = \Sigma_\pi$, see §5.1.12, and then $\mathfrak{R}(\pi) \subseteq \Sigma_\pi$, and also that $\pi \in \mathcal{S}$, so $|\Sigma_\pi| > 0 \Leftrightarrow \Sigma_\pi \neq \emptyset$, see §4.5.8 and §3.2.14, and then $\mathfrak{R}(\pi) = \Sigma_\pi \neq \emptyset$, and therefore $\pi \in \Xi\mathfrak{R}$, see §5.1.11. \diamond

COMMENT For solvable problems, $\Phi\mathfrak{R} \subseteq \Xi\mathfrak{R}$, so they are easier to solve than to resolve. But unsolvable problems, some of them resolved, are impossible to solve!

§5.1.14 DEFINITION We will say that the resources of a resolver are in a set if the capability implemented in the resolver belongs to that set.

COMMENT Now we will construct a series of resolvers \mathfrak{R}_n , from the minimal one that only implements one solution, and then growing naturally step by step. Each resolver will implement just one element out of its resources

NOTATION We will use $\{ \cdot \mathfrak{R}_n \cdot \}$ to refer to the set of all the resolvers of step n .

§5.2 Mechanism

§5.2.1 DEFINITION A mechanism \mathfrak{R}_0 is any resolver that implements one member of \mathbb{S} . We will note $\mathfrak{R}_0[s]$, where $s \in \mathbb{S}$, the mechanism that implements s , that is, $\mathfrak{R}_0[s] = s \in \mathbb{S}$. Then the mechanisms resources are in \mathbb{S} , and $\{ \cdot \mathfrak{R}_0 \cdot \} = \mathbb{S}$.

COMMENT Mechanism $\mathfrak{R}_0[s]$ returns s unconditionally.

COMMENT A mechanism \mathfrak{R}_0 implements a semantic unconditional computation.

§5.2.2 NOTATION As resolutions return sets of elements in \mathbb{S} , to normalize the situation of mechanisms \mathfrak{R}_0 we will use the singleton isomorphism, see §3.2.11, and we will write $\mathfrak{R}_0[\{s\}]$ to mean the singleton $\{s\}$, that is, $\mathfrak{R}_0[\{s\}] = \{\mathfrak{R}_0[s]\} = \{s\} \in 2^{\mathbb{S}}$.

§5.2.3 LEMMA $\forall s \in \mathbb{S}, \Xi \mathfrak{R}_0[\{s\}] = \{\pi \mid P_\pi(s)\}$.

PROOF Just applying the definition of range, see §5.1.11, to the definition of mechanism, we get: $\Xi \mathfrak{R}_0[\{s\}] = \{\pi \mid \mathfrak{R}_0[\{s\}] \subseteq \Sigma_\pi \wedge \mathfrak{R}_0[\{s\}] \neq \emptyset\} = \{\pi \mid \{s\} \subseteq \Sigma_\pi \wedge \{s\} \neq \emptyset\} = \{\pi \mid s \in \Sigma_\pi \wedge \top\} = \{\pi \mid s \in \Sigma_\pi\} = \{\pi \mid P_\pi(s)\}$. \diamond

COMMENT The range of the mechanism $\mathfrak{R}_0[s]$ is the set of problems for which s is a solution.

§5.2.4 LEMMA $\forall s \in \mathbb{S}, \Phi \mathfrak{R}_0[\{s\}] = \{\delta_s\}$.

PROOF Just applying the definition of power, see §5.1.12, to the definition of mechanism, we get: $\Phi \mathfrak{R}_0[\{s\}] = \{\pi \mid \mathfrak{R}_0[\{s\}] = \Sigma_\pi\} = \{\pi \mid \{s\} = \Sigma_\pi\} = \{\delta_s\}$, the last equation because $\Sigma_{\delta_s} = \{s\}$, see §4.3.14. \diamond

COMMENT Mechanism $\mathfrak{R}_0[s]$ only resolves problem δ_s .

§5.2.5 LEMMA Any singleton routine resolution $R_\pi = \{s\}$ can be implemented by the mechanism $\mathfrak{R}_0[R_\pi]$.

PROOF If $R_\pi = \{s\}$, then $R_\pi = \{s\} = \{\mathfrak{R}_0[s]\} = \mathfrak{R}_0[\{s\}] = \mathfrak{R}_0[R_\pi]$. \diamond

COMMENT In theory, we can equal any finite routine resolution to a union of a finite number of mechanisms, $R_\pi = \Sigma_\pi = \bigcup_{s \in \Sigma_\pi} \{s\} = \bigcup_{s \in \Sigma_\pi} \{\mathfrak{R}_0[s]\}$.

§5.2.6 SUMMARY In practice, it only makes sense to implement one solution, as $\mathfrak{R}_0[s]$ does. Without conditional calculations, the mechanism could not control when to apply one result or any of the others, so it would gain nothing implementing more than one.

COMMENT The mechanism is a body capable of one behavior.

EXAMPLE A mechanism can only survive in a specific and very stable environment, as it is the case of some extremophile archaea.

§5.3 Adapter

§5.3.1 DEFINITION An adapter \mathfrak{R}_1 is any resolver that implements one condition on the members of \mathbb{S} . We will note $\mathfrak{R}_1[P_S]$ the adapter that implements P_S , where $P_S \in (\mathbb{S} \rightarrow \mathbb{B})$, that is, $\mathfrak{R}_1[P_S] = P_S \in (\mathbb{S} \rightarrow \mathbb{B})$. Then the adapters resources are in $\mathbb{S} \rightarrow \mathbb{B}$, and $\{\cdot \mathfrak{R}_1 \cdot\} = (\mathbb{S} \rightarrow \mathbb{B})$.

COMMENT An adapter \mathfrak{R}_1 implements a semantic conditional computation.

§5.3.2 LEMMA Each adapter $\mathfrak{R}_1[P_S]$ implements one set of elements of \mathbb{S} .

PROOF Because every predicate P_S defines a set $S = \{s \in \mathbb{S} \mid P_S(s)\} \in 2^{\mathbb{S}}$. The condition P_S is the characteristic function of S , $\forall s \in \mathbb{S}, P_S(s) = [s \in S]$. \diamond

COMMENT We will write $\mathfrak{R}_1[P_S] = \mathfrak{R}_1[S] = S \in 2^{\mathbb{S}}$. Only effectively calculable conditions are implementable, and then adapters can only implement expressible, or recursive, sets, \mathcal{E} . Then, $\mathfrak{R}_1[P_S] = \mathfrak{R}_1[S] = S \in \mathcal{E}$.

§5.3.3 LEMMA Every mechanism \mathfrak{R}_0 is an adapter \mathfrak{R}_1 , that is, $\{\cdot \mathfrak{R}_0 \cdot\} \subset \{\cdot \mathfrak{R}_1 \cdot\}$.

PROOF For each mechanism $\mathfrak{R}_0[s]$, which implements $s \in \mathbb{S}$, there is an adapter $\mathfrak{R}_1[P_{\delta_s}]$, see §4.3.14, that implements the singleton $\{s\} \in (\mathbb{S} \rightarrow \mathbb{B})$. But not every set is a singleton. Summarizing, $\{\cdot \mathfrak{R}_0 \cdot\} = \mathbb{S} \subset (\mathbb{S} \rightarrow \mathbb{B}) = \{\cdot \mathfrak{R}_1 \cdot\}$. \diamond

COMMENT In Cantor's paradise, but out of Turing universes, by the singleton (§3.2.11) and the set (§3.2.8) isomorphisms: $\{\cdot \mathfrak{R}_0 \cdot\} = \mathbb{S} \cong \mathbb{S}^1 \subset 2^{\mathbb{S}} \cong (\mathbb{S} \rightarrow \mathbb{B}) = \{\cdot \mathfrak{R}_1 \cdot\}$.

§5.3.4 LEMMA $\mathfrak{R}_1[S] = \bigcup_{s \in S} \{\mathfrak{R}_0[s]\}$.

PROOF Because $\mathfrak{R}_0[s] = s$, so $\bigcup_{s \in S} \{\mathfrak{R}_0[s]\} = \bigcup_{s \in S} \{s\} = S = \mathfrak{R}_1[S]$. \diamond

COMMENT The results are the same, but not the implementation, because while the adapter $\mathfrak{R}_1[S]$ implements a condition, the union of mechanisms $\bigcup_{s \in S} \{\mathfrak{R}_0[s]\}$ works unconditionally. Thus, the output of the union of mechanisms is independent of any problem, and then the union cannot implement $\mathfrak{R}_1[P_S \wedge P_\pi] = \mathfrak{R}_1[S \cap \Sigma_\pi]$, for example, so it cannot implement any trial, see Theorem §3.3.5.

§5.3.5 LEMMA $\forall S \in 2^S, \forall \pi \in \mathbb{P}, \Xi \mathfrak{R}_1[S \cap \Sigma_\pi] = \{\pi \mid S \cap \Sigma_\pi \neq \emptyset\}$.

PROOF Because $\Xi \mathfrak{R}_1[S \cap \Sigma_\pi] = \{\pi \mid (\mathfrak{R}_1[S \cap \Sigma_\pi] \subseteq \Sigma_\pi) \wedge (\mathfrak{R}_1[S \cap \Sigma_\pi] \neq \emptyset)\} = \{\pi \mid (S \cap \Sigma_\pi \subseteq \Sigma_\pi) \wedge (S \cap \Sigma_\pi \neq \emptyset)\} = \{\pi \mid \top \wedge (S \cap \Sigma_\pi \neq \emptyset)\} = \{\pi \mid S \cap \Sigma_\pi \neq \emptyset\}$. \diamond

COMMENT The range of the adapter $\mathfrak{R}_1[S \cap \Sigma_\pi]$ is the set of problems that have any solution in S . The adapter $\mathfrak{R}_1[S \cap \Sigma_\pi]$ solves any problem such that any of its solutions are in S .

COROLLARY If $S \subset S'$, then $\Xi \mathfrak{R}_1[S \cap \Sigma_\pi] \subset \Xi \mathfrak{R}_1[S' \cap \Sigma_\pi]$.

PROOF In that case, if a solution to a problem is in S , then it is also in S' . But there are also solutions in S' that are not in S . \diamond

§5.3.6 LEMMA If $s \in S$, then $\forall \pi \in \mathbb{P}, \Xi \mathfrak{R}_0[\{s\}] \subseteq \Xi \mathfrak{R}_1[S \cap \Sigma_\pi]$.

PROOF By Lemma §5.2.3, $\forall \pi \in \Xi \mathfrak{R}_0[\{s\}]$, $P_\pi(s)$, that is, $s \in \Sigma_\pi$, so, if $s \in S$, then $S \cap \Sigma_\pi \neq \emptyset$, and therefore $\pi \in \Xi \mathfrak{R}_1[S \cap \Sigma_\pi]$, by Lemma §5.3.5. \diamond

DEFINITION We will call $s \in S$ the adapter condition. If the adapter condition holds, then the adapter $\mathfrak{R}_1[S \cap \Sigma_\pi]$ solves any problem that the mechanism $\mathfrak{R}_0[s]$ solves.

COROLLARY If $\{s\} \subset S$, then $\Xi \mathfrak{R}_0[\{s\}] \subset \Xi \mathfrak{R}_1[S \cap \Sigma_\pi]$.

PROOF Because, if $t \in S$ and $t \neq s$, then $\delta_t \in \Xi \mathfrak{R}_1[S \cap \Sigma_\pi]$ but $\delta_t \notin \Xi \mathfrak{R}_0[\{s\}]$. \diamond

PROPOSITION If $\{s\} \subset S$, then $\Xi \mathfrak{R}_0[\{s\}] \not\subseteq \Xi \mathfrak{R}_1[S]$.

Because $\delta_s \in \Xi \mathfrak{R}_0[\{s\}]$, but $\delta_s \notin \Xi \mathfrak{R}_1[S]$.

§5.3.7 LEMMA $\forall S \in 2^S, \forall \pi \in \mathbb{P}, \Phi \mathfrak{R}_1[S \cap \Sigma_\pi] = 2^S$.

PROOF Because $\Phi \mathfrak{R}_1[S \cap \Sigma_\pi] = \{\pi \mid \mathfrak{R}_1[S \cap \Sigma_\pi] = \Sigma_\pi\} = \{\pi \mid S \cap \Sigma_\pi = \Sigma_\pi\} = \{\pi \mid \Sigma_\pi \subseteq S\} = 2^S$, where the last equality uses the set isomorphism, see §3.2.8. \diamond

COMMENT The power of the adapter $\mathfrak{R}_1[S \cap \Sigma_\pi]$ is the powerset of S . The adapter $\mathfrak{R}_1[S \cap \Sigma_\pi]$ resolves any problem such that all of its solutions are in S .

COROLLARY If $S \subset S'$, then $\Phi \mathfrak{R}_1[S \cap \Sigma_\pi] \subset \Phi \mathfrak{R}_1[S' \cap \Sigma_\pi]$.

PROOF Just because, if $S \subset S'$, then $2^S \subset 2^{S'}$. \diamond

§5.3.8 LEMMA If $s \in S$, then $\forall \pi \in \mathbb{P}, \Phi \mathfrak{R}_0[\{s\}] \subset \Phi \mathfrak{R}_1[S \cap \Sigma_\pi]$.

PROOF Using the set isomorphism, see §3.2.8, $\delta_s = \Sigma_{\delta_s} = \{s\}$, and then, if $s \in S$, $\Phi \mathfrak{R}_0[\{s\}] = \{\delta_s\} = \{\{s\}\} \subset 2^S = \Phi \mathfrak{R}_1[S \cap \Sigma_\pi]$, by Lemmas §5.2.4 and §5.3.7. \diamond

COMMENT If the adapter condition holds, $s \in S$, then the adapter $\mathfrak{R}_1[S \cap \Sigma_\pi]$ resolves any problem that the mechanism $\mathfrak{R}_0[s]$ resolves, and more.

PROPOSITION If $\{s\} \subset S$, then $\Phi \mathfrak{R}_0[\{s\}] \not\subseteq \Phi \mathfrak{R}_1[S]$, because $\delta_s \notin \Phi \mathfrak{R}_1[S] = \{S\}$.

§5.3.9 LEMMA Any effectively calculable trial resolution $T_\pi(S)$ can be implemented by the adapter $\mathfrak{R}_1[S \cap \Sigma_\pi]$.

PROOF $T_\pi(S) = \{s \in S \mid s \in \Sigma_\pi\} = \{s \mid s \in S \wedge s \in \Sigma_\pi\} = \{s \mid P_S(s) \wedge P_\pi(s)\}$. Then $T_\pi(S) \doteq \mathfrak{R}_1[P_S \wedge P_\pi] = \mathfrak{R}_1[S \cap \Sigma_\pi]$. The equality is dotted because, if the trial is not an effectively calculable function, then it cannot be implemented. \diamond

§5.3.10 SUMMARY In practice, an adapter $\mathfrak{R}_1[P_S \wedge P_\pi] = \mathfrak{R}_1[S \cap \Sigma_\pi]$ has a body capable of several behaviors that provides the set S of behaviors. If the current behavior were not satisfying the adapter condition P_π , which is interpreted as an error, then the adapter would change its behavior trying another one in S .

COMMENT The adapter is a body capable of several behaviors, and a governor that selects the current behavior.

EXAMPLE A deciduous tree, which switches its behavior with seasons, is an adapter.

§5.4 Perceiver

§5.4.1 DEFINITION A perceiver \mathfrak{R}_2 is any resolver that implements one transformation of the elements in \mathbb{S} into the elements in \mathbb{S} . We will note $\mathfrak{R}_2[f]$ the perceiver that implements f , where $f \in (\mathbb{S} \rightarrow \mathbb{S})$, that is, $\mathfrak{R}_2[f] = f \in (\mathbb{S} \rightarrow \mathbb{S})$. Then the perceiver resources are in $\mathbb{S} \rightarrow \mathbb{S}$, and $\{\cdot \mathfrak{R}_2 \cdot\} = (\mathbb{S} \rightarrow \mathbb{S})$.

COMMENT From a semantic point of view, a perceiver \mathfrak{R}_2 implements a semantic functional computation. From a syntactic point of view, a perceiver \mathfrak{R}_2 implements a syntactic unconditional computation.

§5.4.2 REMARK Perceivers are to syntax as mechanisms are to semantics.

COMMENT When solutions are functions $\mathbb{S} \rightarrow \mathbb{S}$, then a perceiver does what a mechanism does, which is to return a solution unconditionally. That is, perceivers on metaproblems are as mechanisms on problems. But, perceivers can go further.

COMMENT The perceiver $\mathfrak{R}_2[f]$ implements function f from \mathbb{S} to \mathbb{S} , that is, $f : \mathbb{S} \rightarrow \mathbb{S}$. Then, $\forall s \in \mathbb{S}$, $\mathfrak{R}_2[f](s) = f(s) \in \mathbb{S}$.

§5.4.3 NOTATION By the rewriting rules in §3.1.3, $\mathfrak{R}_2[f](S) = \{ \mathfrak{R}_2[f](s) \mid s \in S \} \in 2^{\mathbb{S}}$. Then $\mathfrak{R}_2[f](S)$ returns a set of solutions, as any well-behaved resolution should do.

COMMENT The perceiver $\mathfrak{R}_2[f](S)$ implements f , meaning that f is hardwired in the perceiver, while S is just data. We will call what is implemented hardware, and what is data software. We write the hardware between brackets, and the software between parentheses. We will assume that coding software costs less than implementing hardware, or, in fewer words, that software is cheaper than hardware

§5.4.4 LEMMA Every adapter \mathfrak{R}_1 is a perceiver \mathfrak{R}_2 , that is, $\{\cdot \mathfrak{R}_1 \cdot\} \subset \{\cdot \mathfrak{R}_2 \cdot\}$.

PROOF Because $\mathbb{B} \subset \mathbb{S}$, and then $(\mathbb{S} \rightarrow \mathbb{B}) \subset (\mathbb{S} \rightarrow \mathbb{S})$. So

$$\{\cdot \mathfrak{R}_1 \cdot\} = (\mathbb{S} \rightarrow \mathbb{B}) \subset (\mathbb{S} \rightarrow \mathbb{S}) = \{\cdot \mathfrak{R}_2 \cdot\}. \diamond$$

COMMENT Each adapter implements one condition $P_S \in (\mathbb{S} \rightarrow \mathbb{B})$. And any condition $P_S \in (\mathbb{S} \rightarrow \mathbb{B})$ is also a function $P_S \in (\mathbb{S} \rightarrow \mathbb{S})$, because $\mathbb{B} \subset \mathbb{S}$. Therefore, for each adapter $\mathfrak{R}_1[P_S]$, which implements condition P_S , there is a perceiver $\mathfrak{R}_2[P_S]$ that implements the function P_S , and then we write $\mathfrak{R}_1[P_S] = P_S = \mathfrak{R}_2[P_S]$.

COMMENT Again, $\mathfrak{R}_1[P_S] = \mathfrak{R}_2[P_S]$ explains that the results are the same, but not the implementation.

§5.4.5 LEMMA $\forall S \in 2^{\mathbb{S}}$, $\mathfrak{R}_1[S] = \mathfrak{R}_2[i](S)$.

PROOF Function $i : \mathbb{S} \rightarrow \mathbb{S}$ is the semantic identity, $i = \mathbf{i}$ see §5.1.10, so $\forall s \in \mathbb{S}$, $i(s) = s$, and $\mathfrak{R}_2[i](S) = \{ \mathfrak{R}_2[i](s) \mid s \in S \} = \{ i(s) \mid s \in S \} = \{ s \mid s \in S \} = S = \mathfrak{R}_1[S]$. \diamond

COMMENT The same perceiver hardware $\mathfrak{R}_2[i]$, just by changing its software, can emulate different adapters: $\mathfrak{R}_2[i](S) = \mathfrak{R}_1[S]$, and $\mathfrak{R}_2[i](S') = \mathfrak{R}_1[S']$. Then the perceiver $\mathfrak{R}_2[i](S)$ is more flexible than the adapter $\mathfrak{R}_1[S]$, because S is hardwired in the adapter, while it is easily replaceable data for the perceiver.

§5.4.6 LEMMA $\forall S \in 2^{\mathbb{S}}, \Xi \mathfrak{R}_1[S] = \Xi \mathfrak{R}_2[i](S)$ and $\Phi \mathfrak{R}_1[S] = \Phi \mathfrak{R}_2[i](S)$.

PROOF Because, by Lemma §5.4.5, $\mathfrak{R}_1[S] = \mathfrak{R}_2[i](S)$. \diamond

DEFINITION The perceiver condition is satisfied if it implements the semantic identity i .

COMMENT If the perceiver condition holds, then the perceiver $\mathfrak{R}_2[i](S)$ solves any problem solved by the adapter $\mathfrak{R}_1[S]$, and the perceiver $\mathfrak{R}_2[i](S)$ resolves any problem resolved by the adapter $\mathfrak{R}_1[S]$.

REMARK Semantic identity i is the ideal for perception.

§5.4.7 COROLLARY $\Xi \mathfrak{R}_2[i](S \cap \Sigma_\pi) = \Xi \mathfrak{R}_1[S \cap \Sigma_\pi]$ and $\Phi \mathfrak{R}_2[i](S \cap \Sigma_\pi) = \Phi \mathfrak{R}_1[S \cap \Sigma_\pi]$.

PROOF By Lemma §5.4.6. \diamond

COMMENT The same perceiver hardware $\mathfrak{R}_2[i]$ can be tuned to a different trial just by changing its software, from $\mathfrak{R}_2[i](S \cap \Sigma_\pi)$ to $\mathfrak{R}_2[i](S' \cap \Sigma_\rho)$, for example.

PROPOSITION If $S \subset S'$, then $\Xi \mathfrak{R}_2[i](S \cap \Sigma_\pi) \subset \Xi \mathfrak{R}_2[i](S' \cap \Sigma_\pi)$, and $\Phi \mathfrak{R}_2[i](S \cap \Sigma_\pi) \subset \Phi \mathfrak{R}_2[i](S' \cap \Sigma_\pi)$, by Lemma §5.4.5 and corollaries to Lemmas §5.3.5 and §5.3.7.

§5.4.8 DEFINITION A function on sets $F : 2^{\mathbb{S}} \rightarrow 2^{\mathbb{S}}$ is elementable if it exists an effectively calculable function $f : \mathbb{S} \rightarrow \mathbb{S}$ such that $\forall S, F(S) = \{ f(s) \mid s \in S \}$.

COMMENT We write $F(S) = f(S)$, by the rules in §3.1.3. Note the three requirements: that f is a semantic function, $f : \mathbb{S} \rightarrow \mathbb{S}$, that f is effectively calculable, and that $F(S) = f(S)$.

PROPOSITION Set identity $I : 2^{\mathbb{S}} \rightarrow 2^{\mathbb{S}} \mid \forall S \in 2^{\mathbb{S}}, I(S) = S$, is elementable by semantic identity i , because $\forall S \in 2^{\mathbb{S}}, i(S) = \{ i(s) \mid s \in S \} = \{ s \mid s \in S \} = S = I(S)$.

§5.4.9 LEMMA Any analogy resolution $A \circ T_{A\pi}(S) \circ \mathcal{T}_A$ can be implemented by the tri-perceiver $\mathfrak{R}_2[\mathcal{T}_a](\mathfrak{R}_2[i](S \cap \mathfrak{R}_2[a](\Sigma_\pi)))$, if A is elementable by a , and \mathcal{T}_A by \mathcal{T}_a .

PROOF An analogy resolution is $A \circ T_{A\pi}(S) \circ \mathcal{T}_A$. Both A and \mathcal{T}_A are functions from sets to sets, $\mathcal{T}_A : 2^{\mathbb{S}} \rightarrow 2^{\mathbb{S}}$ and $A : (\mathbb{P} \rightarrow \mathbb{P}) = (2^{\mathbb{S}} \rightarrow 2^{\mathbb{S}})$, so if both A and \mathcal{T}_A are elementable, then a perceiver can implement them. We have a and \mathcal{T}_a , which are both semantic functions such that $A(S) = a(S)$, and $\mathcal{T}_A(S) = \mathcal{T}_a(S)$. Then $P_{a\pi} = a(P_\pi) = \mathfrak{R}_2[a](P_\pi) = \mathfrak{R}_2[a](\Sigma_\pi)$ implements the first third, $T_{a\pi}(S) = \mathfrak{R}_1[P_S \wedge P_{a\pi}] = \mathfrak{R}_1[P_S \wedge \mathfrak{R}_2[a](P_\pi)] = \mathfrak{R}_1[S \cap \mathfrak{R}_2[a](\Sigma_\pi)]$ implements the second third, see §5.3.9, and using §5.4.5, $\mathfrak{R}_2[\mathcal{T}_a](\mathfrak{R}_2[i](S \cap \mathfrak{R}_2[a](\Sigma_\pi)))$ implements the whole analogy resolution $a \circ T_{a\pi}(S) \circ \mathcal{T}_a$. \diamond

COROLLARY Identity analogy $I \circ T_{I\pi}(S) \circ \mathcal{T}_I$ can be implemented by the bi-perceiver $\mathfrak{R}_2[i](S \cap \mathfrak{R}_2[i](\Sigma_\pi))$, which uses the identity perceiver $\mathfrak{R}_2[i]$ twice.

PROOF $\mathfrak{R}_2[\mathcal{T}_i](\mathfrak{R}_2[i](S \cap \mathfrak{R}_2[i](\Sigma_\pi))) = \mathfrak{R}_2[i](\mathfrak{R}_2[i](S \cap \mathfrak{R}_2[i](\Sigma_\pi))) = \mathfrak{R}_2[i](S \cap \mathfrak{R}_2[i](\Sigma_\pi))$, because set identity I is elementable by semantic identity i , and $\mathcal{T}_I = I$, so it is also elementable by $\mathcal{T}_i = i$, and $i \circ i = i$. \diamond

§5.4.10 SUMMARY While an adapter uses a trial and error resolution, and this means that error is part of the usual procedure, a perceiver executes the trial and error inside itself. If the analogy provides a good model, then the internal trial is as good as the external one, with the advantage that the errors are only simulated errors. More to the point, if the problem the resolver faces is the survival problem, then the adapter errors are literally death errors, or at least pain, while the perceiver errors are just mental previsions of what not to do. See that, if the perceiver implements the identity analogy, as $\mathfrak{R}_2[i]$ does, then the model is good, because the internal problem is equal to the external one. And the perceiver $\mathfrak{R}_2[i]$ is more flexible than the adapter.

COMMENT The perceiver is a body capable of several behaviors, a governor that selects the current behavior, and a simulator that internalizes behaviors.

EXAMPLE The perceiver governor determines what to do based upon an internal interpretation. According to [Lettvin et al. \(1959\)](#), a frog is a perceiver that uses an internal routine. Frog's i is such that any dark point that moves rapidly in its field of vision is a fly which it will try to eat.

§5.5 Learner

§5.5.1 DEFINITION A learner \mathfrak{R}_3 is any resolver that implements one condition on the members of $\mathbb{S} \rightarrow \mathbb{S}$. We will note $\mathfrak{R}_3[P_F]$ the learner that implements P_F , where $P_F \in ((\mathbb{S} \rightarrow \mathbb{S}) \rightarrow \mathbb{B})$, that is, $\mathfrak{R}_1[P_F] = P_F \in ((\mathbb{S} \rightarrow \mathbb{S}) \rightarrow \mathbb{B})$. Then the learners resources are in $(\mathbb{S} \rightarrow \mathbb{S}) \rightarrow \mathbb{B}$, and $\{\cdot \mathfrak{R}_3 \cdot\} = ((\mathbb{S} \rightarrow \mathbb{S}) \rightarrow \mathbb{B})$.

COMMENT A learner \mathfrak{R}_3 implements a syntactic conditional computation.

§5.5.2 REMARK Learners are to syntax as adapters are to semantics.

COMMENT When solutions are functions $\mathbb{S} \rightarrow \mathbb{S}$, then a learner does what an adapter does, which is to return a predicate on solutions. That is, learners on metaproblems are as adapters on problems. But, learners can go further.

§5.5.3 LEMMA Each learner $\mathfrak{R}_3[P_F]$ implements one set of members of $(\mathbb{S} \rightarrow \mathbb{S})$.

PROOF Because every predicate $P_F : (\mathbb{S} \rightarrow \mathbb{S}) \rightarrow \mathbb{B}$ defines a set

$F = \{f \in (\mathbb{S} \rightarrow \mathbb{S}) \mid P_F(f)\} \in 2^{\mathbb{S} \rightarrow \mathbb{S}}$. The condition P_F is the characteristic function of F , $\forall f \in (\mathbb{S} \rightarrow \mathbb{S})$, $P_F(f) = [f \in F]$. \diamond

COMMENT We will write $\mathfrak{R}_3[P_F] = \mathfrak{R}_3[F] = F \in 2^{\mathbb{S} \rightarrow \mathbb{S}}$.

COMMENT The learner $\mathfrak{R}_3[F]$ implements $F \in 2^{\mathbb{S} \rightarrow \mathbb{S}}$. So $\forall s \in \mathbb{S}$, $\mathfrak{R}_3[F](s) = F(s) \in 2^{\mathbb{S}}$, because $F(s) = \{f(s) \mid f \in F\}$, by the rewriting rules in [§3.1.3](#), and then $\mathfrak{R}_3[F](s)$ returns a set of solutions, as any well-behaved resolution should do. Also, by the same rules, $\mathfrak{R}_3[F](S) = F(S) = \{f(s) \mid s \in S \times f \in F\} \in 2^{\mathbb{S}}$.

§5.5.4 LEMMA Every perceiver \mathfrak{R}_2 is a learner \mathfrak{R}_3 , that is, $\{\cdot \mathfrak{R}_2 \cdot\} \subset \{\cdot \mathfrak{R}_3 \cdot\}$.

PROOF For each perceiver $\mathfrak{R}_2[f]$, which implements $f \in (\mathbb{S} \rightarrow \mathbb{S})$, there is a learner $\mathfrak{R}_3[P_{\delta_f}]$, see [§4.3.14](#), that implements the singleton $\{f\} \in ((\mathbb{S} \rightarrow \mathbb{S}) \rightarrow \mathbb{B})$. But not every set is a singleton. Then, $\{\cdot \mathfrak{R}_2 \cdot\} = (\mathbb{S} \rightarrow \mathbb{S}) \subset ((\mathbb{S} \rightarrow \mathbb{S}) \rightarrow \mathbb{B}) = \{\cdot \mathfrak{R}_3 \cdot\}$. \diamond

§5.5.5 LEMMA $\mathfrak{R}_3[F] = \bigcup_{f \in F} \{\mathfrak{R}_2[f]\}$.

PROOF Because $\mathfrak{R}_2[f] = f$, so $\bigcup_{f \in F} \{\mathfrak{R}_2[f]\} = \bigcup_{f \in F} \{f\} = F = \mathfrak{R}_3[F]$. \diamond

COMMENT Again, the results are the same, but not the implementation. The union of perceivers $\bigcup_{f \in F} \{\mathfrak{R}_2[f]\}$ cannot select a function to use, so it cannot implement any meta-trial, as $\mathfrak{R}_3[R \cap \Sigma_{\Pi\pi}]$ does, see [§5.5.11](#).

COMMENT These are not sets of solutions, but sets of semantic functions.

§5.5.6 LEMMA $\forall f \in F, \forall S \in 2^{\mathbb{S}}, \mathfrak{R}_2[f](S) \subseteq \mathfrak{R}_3[F](S)$.

PROOF If $f \in F$, then $f(S) = \{f(s) \mid s \in S\} \subseteq \{f'(s) \mid s \in S \times f' \in F\} = F(S)$. \diamond

§5.5.7 NOTATION We will rewrite $\mathfrak{R}_1[\mathfrak{R}_3[F](S) \cap \Sigma_\pi]$ as $\mathfrak{R}_3[F](S \cap \Sigma_\pi)$.

COMMENT We can write $\mathfrak{R}_1[\mathfrak{R}_3[F](S) \cap \Sigma_\pi] = \mathfrak{R}_3[F](S \cap \Sigma_\pi)$, because any learner can implement semantic conditions, that is, because $\{\cdot \mathfrak{R}_1 \cdot\} \subset \{\cdot \mathfrak{R}_3 \cdot\}$.

§5.5.8 LEMMA If $f \in F$, then $\forall S \in 2^{\mathbb{S}}, \forall \pi \in \mathbb{P}, \Xi \mathfrak{R}_2[f](S) \subseteq \Xi \mathfrak{R}_3[F](S \cap \Sigma_\pi)$.

PROOF Firstly see that, if $f \in F$, then $f(S) \subseteq F(S)$, by §3.1.3, so, $f(S) \cap \Sigma_\pi \subseteq F(S) \cap \Sigma_\pi$. Secondly see that $\forall \pi \in \Xi \mathfrak{R}_2[f](S), f(S) \cap \Sigma_\pi = f(S) \neq \emptyset$. This is because $\Xi \mathfrak{R}_2[f](S) = \{\pi \mid f(S) \subseteq \Sigma_\pi \wedge f(S) \neq \emptyset\}$. Now, taking both together, $\forall \pi \in \Xi \mathfrak{R}_2[f](S), \emptyset \neq f(S) = f(S) \cap \Sigma_\pi \subseteq F(S) \cap \Sigma_\pi$, so for these π , $F(S) \cap \Sigma_\pi \neq \emptyset$. Then these $\pi \in \{\pi \mid F(S) \cap \Sigma_\pi \subseteq \Sigma_\pi \wedge F(S) \cap \Sigma_\pi \neq \emptyset\} = \Xi \mathfrak{R}_1[\mathfrak{R}_3[F](S) \cap \Sigma_\pi]$, because $F(S) \cap \Sigma_\pi \subseteq \Sigma_\pi$ is always true. \diamond

DEFINITION We will call $f \in F$ the learner condition. If the learner condition holds, then the learner $\mathfrak{R}_3[F](S \cap \Sigma_\pi)$ solves any problem that the perceiver $\mathfrak{R}_2[f](S)$ solves.

§5.5.9 LEMMA If $f \in F$, then $\forall S \in 2^{\mathbb{S}}, \forall \pi \in \mathbb{P}, \Phi \mathfrak{R}_2[f](S) \subseteq \Phi \mathfrak{R}_3[F](S \cap \Sigma_\pi)$.

PROOF If $f \in F$, then $f(S) \subseteq F(S)$, see §3.1.3, so, $f(S) \cap \Sigma_\pi \subseteq F(S) \cap \Sigma_\pi$. Now $\forall \pi \in \Phi \mathfrak{R}_2[f](S), f(S) = \Sigma_\pi$, and then for these π , $\Sigma_\pi = f(S) \cap \Sigma_\pi \subseteq F(S) \cap \Sigma_\pi \subseteq \Sigma_\pi$. Therefore, for these π , $F(S) \cap \Sigma_\pi = \Sigma_\pi$, and $\Phi \mathfrak{R}_2[f](S) \subseteq \{\pi \mid F(S) \cap \Sigma_\pi = \Sigma_\pi\} = \Phi \mathfrak{R}_1[\mathfrak{R}_3[F](S) \cap \Sigma_\pi]$. \diamond

COMMENT If the learner condition holds, $f \in F$, then the learner $\mathfrak{R}_3[F](S \cap \Sigma_\pi)$ resolves any problem that the perceiver $\mathfrak{R}_2[f](S)$ resolves.

§5.5.10 COROLLARY In particular, if $i \in R$, then $\Xi \mathfrak{R}_2[i](S \cap \Sigma_\pi) \subseteq \Xi \mathfrak{R}_3[R](S \cap \Sigma_\pi)$ and $\Phi \mathfrak{R}_2[i](S \cap \Sigma_\pi) \subseteq \Phi \mathfrak{R}_3[R](S \cap \Sigma_\pi)$.

PROOF By Lemmas §5.5.8 and §5.5.9. See that $\Xi \mathfrak{R}_3[R](S \cap \Sigma_\pi \cap \Sigma_\pi) \subseteq \Xi \mathfrak{R}_3[R](S \cap \Sigma_\pi)$, and $\Phi \mathfrak{R}_3[R](S \cap \Sigma_\pi \cap \Sigma_\pi) \subseteq \Phi \mathfrak{R}_3[R](S \cap \Sigma_\pi)$, because $\mathfrak{R}_3[R](S \cap \Sigma_\pi) \subseteq \mathfrak{R}_3[R](S)$, so corollaries to Lemmas §5.3.5 and §5.3.7 apply (the equal case is trivial). \diamond

§5.5.11 LEMMA Any meta-trial resolution $T_{\Pi\pi}(R)$ can be implemented by the learner $\mathfrak{R}_3[R \cap \Sigma_{\Pi\pi}]$, if P_R and $P_{\Pi\pi}$ are elementable.

COMMENT The diagram for the meta-trial, or trial of the metaproblem, see §3.6.9, is:

$$\pi \xrightarrow{\Pi} \Pi\pi \xrightarrow{T_{\Pi\pi}(R)} \Sigma_{\Pi\pi} \xrightarrow{\mathcal{T}_\Pi} \Sigma_\pi .$$

COMMENT A learner solves metaproblems by trial, as an adapter solves problems by trial. The following correlations stand: $\mathfrak{R}_3 \leftrightarrow \mathfrak{R}_1$, $\Pi\pi \leftrightarrow \pi$, $R \leftrightarrow S$, and then $\mathfrak{R}_3[R \cap \Sigma_{\Pi\pi}] \leftrightarrow \mathfrak{R}_1[S \cap \Sigma_\pi]$. Therefore, $\mathfrak{R}_3[R \cap \Sigma_{\Pi\pi}]$ compares to $\mathfrak{R}_2[i]$, where $i \in R$, as $\mathfrak{R}_1[S \cap \Sigma_\pi]$ compares to $\mathfrak{R}_0[s]$, where $s \in S$.

PROOF $T_{\Pi\pi}(R) = \{r \in R \mid r \in \Sigma_{\Pi\pi}\} = \{r \mid r \in R \wedge r \in \Sigma_{\Pi\pi}\} = \{r \mid P_R(r) \wedge P_{\Pi\pi}(r)\}$. In the meta-trial $T_{\Pi\pi}(R)$, R is a set of resolutions, where $\mathbb{R} = (\mathbb{P} \rightarrow 2^{\mathbb{S}}) = (2^{\mathbb{S}} \rightarrow 2^{\mathbb{S}})$, that is, $P_R : (2^{\mathbb{S}} \rightarrow 2^{\mathbb{S}}) \rightarrow \mathbb{B}$, and the condition of the metaproblem $\Pi\pi$ is also $P_{\Pi\pi} : \mathbb{R} \rightarrow \mathbb{B} = (\mathbb{P} \rightarrow 2^{\mathbb{S}}) \rightarrow \mathbb{B} = (2^{\mathbb{S}} \rightarrow 2^{\mathbb{S}}) \rightarrow \mathbb{B}$. So if both P_R and $P_{\Pi\pi}$ are elementable by \wp_R and $\wp_{\Pi\pi}$, then both of them, and its conjunction, are implementable. Then $T_{\Pi\pi}(R) = \{r \mid P_R(r) \wedge P_{\Pi\pi}(r)\} \doteq \mathfrak{R}_3[\wp_R \wedge \wp_{\Pi\pi}] = \mathfrak{R}_3[R \cap \Sigma_{\Pi\pi}]$. \diamond

§5.5.12 SUMMARY Perceiver success depends crucially on the analogy, that is, on how much the analogy resembles the identity i . And a learner can adapt the analogy to the problem it is facing, because the learner $\mathfrak{R}_3[R]$ implements a set of functions R from which it can select another analogy when the current one fails. Adapting the analogy is also known as modeling. So a learner can apply different analogies, but a learner can also apply a routine if it knows a solution, because the routine is more efficient, or a trial, when the model is not good enough or too pessimistic.

COMMENT The learner is a body capable of several behaviors, a governor that selects the current behavior, a simulator that internalizes behaviors, and a modeler that adjusts the model used by the simulator.

EXAMPLE Where there is modeling and simulation there is learning, because enhancing the model prevents repeating errors. A dog is a learner.

§5.6 Subject

§5.6.1 DEFINITION A subject \mathfrak{R}_4 is any resolver that implements one transformation of the elements in $\mathbb{S} \rightarrow \mathbb{S}$ into the elements in $\mathbb{S} \rightarrow \mathbb{S}$. We will note $\mathfrak{R}_4[\mathfrak{f}]$ the subject that implements \mathfrak{f} , where $\mathfrak{f} \in ((\mathbb{S} \rightarrow \mathbb{S}) \rightarrow (\mathbb{S} \rightarrow \mathbb{S}))$, that is, $\mathfrak{R}_4[\mathfrak{f}] = \mathfrak{f} \in ((\mathbb{S} \rightarrow \mathbb{S}) \rightarrow (\mathbb{S} \rightarrow \mathbb{S}))$. Then the subject resources are in $(\mathbb{S} \rightarrow \mathbb{S}) \rightarrow (\mathbb{S} \rightarrow \mathbb{S})$, and $\{\cdot \mathfrak{R}_4 \cdot\} = ((\mathbb{S} \rightarrow \mathbb{S}) \rightarrow (\mathbb{S} \rightarrow \mathbb{S}))$. A subject can also be called a reasoner.

COMMENT A subject \mathfrak{R}_4 implements a syntactic functional computation.

§5.6.2 REMARK Subjects are to syntax as perceivers are to semantics.

COMMENT When solutions are functions $\mathbb{S} \rightarrow \mathbb{S}$, then a subject does what a perceiver does, which is to return a function on solutions to solutions. That is, subjects on metaproblems are as perceivers on problems. But, subjects can go further.

COMMENT The subject $\mathfrak{R}_4[\mathfrak{f}]$ implements function \mathfrak{f} from $\mathbb{S} \rightarrow \mathbb{S}$ to $\mathbb{S} \rightarrow \mathbb{S}$, that is, $\mathfrak{f} : (\mathbb{S} \rightarrow \mathbb{S}) \rightarrow (\mathbb{S} \rightarrow \mathbb{S})$. Then, $\forall f \in (\mathbb{S} \rightarrow \mathbb{S}), \mathfrak{R}_4[\mathfrak{f}](f) = \mathfrak{f}(f) \in (\mathbb{S} \rightarrow \mathbb{S})$.

§5.6.3 NOTATION As resolutions return sets of elements in \mathbb{S} , to normalize the situation of subjects, for which $\mathfrak{R}_4[\mathfrak{f}](f)(s) \in \mathbb{S}$, we will use the rewriting rules in §3.1.3 to get $\mathfrak{R}_4[\mathfrak{f}](F)(S) = \{ \mathfrak{R}_4[\mathfrak{f}](f)(s) \mid s \in S \times f \in F \} \in 2^{\mathbb{S}}$.

COMMENT Subject $\mathfrak{R}_4[\mathfrak{f}](F)(S)$ has two software levels: semantics (S) and syntax (F).

§5.6.4 LEMMA Every learner \mathfrak{R}_3 is a subject \mathfrak{R}_4 , that is, $\{\cdot \mathfrak{R}_3 \cdot\} \subset \{\cdot \mathfrak{R}_4 \cdot\}$.

PROOF First we define set $\mathfrak{B} = \{K_{\top}, K_{\perp}\}$, both functions $\mathbb{S} \rightarrow \mathbb{S}$, see §5.1.8. Next we define the natural isomorphism between \mathbb{B} and \mathfrak{B} , mapping \top to the function that always returns \top , which is $K_{\top} = P_{\tau}$, and \perp to the function that always returns \perp , which is $K_{\perp} = P_{\bar{\tau}}$, see §3.1.11. And so $\mathbb{B} \Leftrightarrow \mathfrak{B} : \top \Leftrightarrow K_{\top}, \perp \Leftrightarrow K_{\perp}$, and $\mathbb{B} \cong \mathfrak{B} \subset (\mathbb{S} \rightarrow \mathbb{S})$. Then, $((\mathbb{S} \rightarrow \mathbb{S}) \rightarrow \mathbb{B}) \subset ((\mathbb{S} \rightarrow \mathbb{S}) \rightarrow (\mathbb{S} \rightarrow \mathbb{S}))$, and therefore $\{\cdot \mathfrak{R}_3 \cdot\} = ((\mathbb{S} \rightarrow \mathbb{S}) \rightarrow \mathbb{B}) \subset ((\mathbb{S} \rightarrow \mathbb{S}) \rightarrow (\mathbb{S} \rightarrow \mathbb{S})) = \{\cdot \mathfrak{R}_4 \cdot\}$. \diamond

COMMENT Each learner implements one condition $P_F \in ((\mathbb{S} \rightarrow \mathbb{S}) \rightarrow \mathbb{B})$. And any condition on functions $P_F \in ((\mathbb{S} \rightarrow \mathbb{S}) \rightarrow \mathbb{B})$ is also a function on functions to functions $P_F \in ((\mathbb{S} \rightarrow \mathbb{S}) \rightarrow (\mathbb{S} \rightarrow \mathbb{S}))$, because $\mathbb{B} \subset (\mathbb{S} \rightarrow \mathbb{S})$. Therefore, for each learner $\mathfrak{R}_3[P_F]$, which implements condition P_F , there is a subject $\mathfrak{R}_4[P_F]$ that implements the function P_F , and then we write $\mathfrak{R}_3[P_F] = P_F = \mathfrak{R}_4[P_F]$.

COMMENT Again, $\mathfrak{R}_3[P_F] = \mathfrak{R}_4[P_F]$ explains that the results are the same, but not the implementation.

§5.6.5 LEMMA $\forall F \in 2^{\mathbb{S} \rightarrow \mathbb{S}}, \forall S \in 2^{\mathbb{S}}, \mathfrak{R}_3[F](S) = \mathfrak{R}_4[\mathfrak{u}](F)(S)$.

COMMENT Function \mathfrak{u} is the identity for programs, see §4.2.3, or functional identity, or evaluation, see §4.3.9. Function \mathfrak{u} is syntactic because it is not restricted to semantic objects, see §5.1.10. Syntactic function \mathfrak{u} is equivalent to λ -calculus $I = (\lambda x.x)$, see §3.4.9: $\forall f \in \mathbb{S} \rightarrow \mathbb{S}, \mathfrak{u}(f) = f$ and $\forall s \in \mathbb{S}, \mathfrak{u}(f)(s) = f(s)$.

PROOF By the rewriting rules in §3.1.3, $\mathfrak{R}_4[\mathfrak{u}](F)(S) = \{ \mathfrak{R}_4[\mathfrak{u}](f)(s) \mid s \in S \times f \in F \} = \{ \mathfrak{u}(f)(s) \mid s \in S \times f \in F \} = \{ f(s) \mid s \in S \times f \in F \} = F(S) = \mathfrak{R}_3[F](S)$. \diamond

COROLLARY $\forall F \in 2^{\mathbb{S} \rightarrow \mathbb{S}}, \mathfrak{R}_3[F] = \mathfrak{R}_4[\mathfrak{u}](F)$.

§5.6.6 LEMMA $\forall F \in 2^{\mathbb{S} \rightarrow \mathbb{S}}, \forall S \in 2^{\mathbb{S}}, \Xi \mathfrak{R}_3[F](S) = \Xi \mathfrak{R}_4[\mathbf{u}](F)(S)$ and $\Phi \mathfrak{R}_3[F](S) = \Phi \mathfrak{R}_4[\mathbf{u}](F)(S)$.

PROOF Because, by Lemma §5.6.5, $\mathfrak{R}_3[F](S) = \mathfrak{R}_4[\mathbf{u}](F)(S)$. \diamond

DEFINITION The subject condition is satisfied if it implements the functional identity \mathbf{u} .

COMMENT If the subject condition holds, then the subject $\mathfrak{R}_4[\mathbf{u}](F)(S)$ solves any problem solved by the learner $\mathfrak{R}_3[F](S)$, and also the subject $\mathfrak{R}_4[\mathbf{u}](F)(S)$ resolves any problem resolved by the learner $\mathfrak{R}_3[F](S)$.

REMARK Functional identity \mathbf{u} is the ideal for reason.

§5.6.7 COROLLARY In particular, $\Xi \mathfrak{R}_4[\mathbf{u}](R)(S \cap \Sigma_\pi) = \Xi \mathfrak{R}_3[R](S \cap \Sigma_\pi)$ and $\Phi \mathfrak{R}_4[\mathbf{u}](R)(S \cap \Sigma_\pi) = \Phi \mathfrak{R}_3[R](S \cap \Sigma_\pi)$.

PROOF By Lemma §5.6.6. \diamond

COMMENT Subject $\mathfrak{R}_4[\mathbf{u}](R)(S \cap \Sigma_\pi)$ is more flexible than learner $\mathfrak{R}_3[R](S \cap \Sigma_\pi)$, because R is software for the subject while it is hardware in the learner, and software is cheaper than hardware, see §5.4.3.

§5.6.8 THEOREM Subject $\mathfrak{R}_4[\mathbf{u}]$ is a full resolution machine.

PROOF By Theorem §4.2.1 and Lemma §4.2.3, $\mathfrak{R}_4[\mathbf{u}] = \mathbf{u} = \mathbf{c}(\mathcal{U})$, so using the program isomorphism, see §4.1.12, $\mathfrak{R}_4[\mathbf{u}] = \mathcal{U}$, which is a Turing complete device, and therefore is a full resolution machine, by Theorem §4.4.2. \diamond

§5.6.9 LEMMA Any effectively calculable resolution \mathfrak{R} can be implemented by the subject $\mathfrak{R}_4[\mathbf{u}]$.

PROOF By Theorem §5.6.8. \diamond

COROLLARY Any effectively calculable meta-analogy resolution $\mathcal{A} \circ T_{\text{API}\pi}(R) \circ \mathcal{T}_{\mathcal{A}}$ can be implemented by the subject $\mathfrak{R}_4[\mathbf{u}]$, including meta-resolving, see §3.5.11.

§5.6.10 SUMMARY The subject, by internalizing metaproblems, prevents meta-errors, that is, the subject can test internally a resolution before executing it. The subject is also more flexible than the learner, because subject modeling is done in software, instead of in hardware. And subject $\mathfrak{R}_4[\mathbf{u}]$ can reason about any model. This means that subject $\mathfrak{R}_4[\mathbf{u}]$ is a resolver that can calculate solutions, but also problems and resolutions without limits; it can represent the problem it is facing to itself, and it can represent itself to itself. In this sense, the subject $\mathfrak{R}_4[\mathbf{u}]$ is conscious.

COMMENT The subject is a body capable of several behaviors, a governor that selects the current behavior, a simulator that internalizes behaviors, a modeler that adjusts the model used by the simulator, and a reason that internalizes resolutions.

EXAMPLE It seems that only our species, *Homo sapiens*, is Turing complete. We deal with the evolution to Turing completeness and its relation to language in Casares (2016b).

§5.7 Resolvers Hierarchy

§5.7.1 THEOREM There is a hierarchy of resolvers:

$$\{\cdot \mathfrak{R}_0 \cdot\} \subset \{\cdot \mathfrak{R}_1 \cdot\} \subset \{\cdot \mathfrak{R}_2 \cdot\} \subset \{\cdot \mathfrak{R}_3 \cdot\} \subset \{\cdot \mathfrak{R}_4 \cdot\}.$$

PROOF Because $\mathbb{S} \subset (\mathbb{S} \rightarrow \mathbb{B}) \subset (\mathbb{S} \rightarrow \mathbb{S}) \subset ((\mathbb{S} \rightarrow \mathbb{S}) \rightarrow \mathbb{B}) \subset ((\mathbb{S} \rightarrow \mathbb{S}) \rightarrow (\mathbb{S} \rightarrow \mathbb{S}))$, by Lemmas §5.3.3, §5.4.4, §5.5.4, and §5.6.4. There are not more types of resolvers, because there is not a resolver more capable than $\mathfrak{R}_4[\mathbf{u}] = \mathcal{U}$, by Theorems §5.6.8 and §4.3.4. \diamond

§5.7.2 SUMMARY This table groups concepts closely related from problem theory, as trial, computing theory, as adapter \mathfrak{R}_1 , and set theory, as $S \in \mathbb{S} \rightarrow \mathbb{B}$.

| | Semantics | Syntax | |
|------|--|--|----------|
| one | Routine Mechanism \mathfrak{R}_0 $s \in \mathbb{S}$ | Meta-routine Perceiver \mathfrak{R}_2 $f \in (\mathbb{S} \rightarrow \mathbb{S})$ | element |
| some | Trial Adapter \mathfrak{R}_1 $S \in \mathbb{S} \rightarrow \mathbb{B}$ | Meta-trial Learner \mathfrak{R}_3 $F \in (\mathbb{S} \rightarrow \mathbb{S}) \rightarrow \mathbb{B}$ | set |
| any | Analogy Perceiver \mathfrak{R}_2 $f \in \mathbb{S} \rightarrow \mathbb{S}$ | Meta-analogy Subject \mathfrak{R}_4 $\mathfrak{f} \in (\mathbb{S} \rightarrow \mathbb{S}) \rightarrow (\mathbb{S} \rightarrow \mathbb{S})$ | function |
| | \mathbb{S} Elements | $\mathbb{S} \rightarrow \mathbb{S}$ Functions | |

COMMENT A perceiver is a syntactic mechanism. A learner is a syntactic adapter. A subject is a syntactic perceiver. A subject is a syntactic² mechanism.

§5.7.3 THEOREM The problem theory is complete.

PROOF Aside from definitions, the problem theory posits that there are three ways to resolve a problem: routine, trial, and analogy; see §2.3. Adding the metaproblem of the problem, we get five ways to resolve a problem and its metaproblem, which are the basic three plus meta-trial and meta-analogy, see §3.6.8. For each way there is a resolver, see Lemmas §5.2.5, §5.3.9, §5.4.9, §5.5.11, and §5.6.9, and the resources of each resolver are in a series of mathematical objects of increasing generality that covers everything until syntactic functions, see §5.7.1 and §5.7.2. Now, to execute meta-analogies, $2^{2^{\mathbb{S}} \rightarrow 2^{\mathbb{S}}} \rightarrow 2^{2^{\mathbb{S}} \rightarrow 2^{\mathbb{S}}}$, see §3.6.9, or at least the elementable ones, see §5.4.8, we need subjects, which implement syntactic functions $(\mathbb{S} \rightarrow \mathbb{S}) \rightarrow (\mathbb{S} \rightarrow \mathbb{S})$. And there is a subject that is a Turing complete device, $\mathfrak{R}_4[u]$, see §5.6.8, so it has the maximum computing power, see §4.3.4, and then the maximum resolving power, see §4.4.2. This means that there are not more resolvers beyond the subject, and therefore that the series is complete, and then that the problem theory covers everything and is complete. \diamond

COMMENT It also means that no more resolutions are needed, although we could do without routine, for example, by using Theorem §3.3.7, and then reducing routines to trials. Nevertheless, a routine is not a trial, because a semantic element is not a semantic set, or because a mechanism implementing a routine is not an adapter implementing a trial, see comment to Lemma §5.3.4.

COMMENT This theorem is true if the Turing's thesis is true, see §4.3.1. Conversely, if this theorem is true, then 'what is effectively calculable to resolve problems is computable'.

§5.7.4 REMARK Provided that a bigger range means more survival opportunities, that software is cheaper than hardware, that the adapter, the perceiver, the learner, and the subject conditions are satisfied in some environments, and that in each step the increasing of complexity was overcome by its fitness, then an evolution of resolvers—mechanism to adapter to perceiver to learner to subject—should follow.

COMMENT Although depending on conditions, see Lemmas §5.3.6, §5.4.6, §5.5.8, and §5.6.6, the evolution of resolvers is directed, and its final singularity is the Turing complete subject $\mathfrak{R}_4[\mathbf{u}]$.

COMMENT In detail, the strictest evolution of resolvers is: $\Xi\mathfrak{R}_0[s] \xrightarrow{\{s\} \subset S} \Xi\mathfrak{R}_1[S \cap \Sigma_\pi] \xrightarrow{S \subset S'} \Xi\mathfrak{R}_2[i](S' \cap \Sigma_\pi) \xrightarrow{\{i\} \subset R} \Xi\mathfrak{R}_3[R](S' \cap \Sigma_\pi) \xrightarrow{R \subset R'} \Xi\mathfrak{R}_4[\mathbf{u}](R')(S' \cap \Sigma_\pi)$.

§5.7.5 THESIS We are the result of an evolution of resolvers of the survival problem.

ARGUMENT The resolvers hierarchy suggests an evolution of resolvers of the survival problem, see §5.7.4. And lacking of better explanations, that we are Turing complete resolvers, that is, subjects $\mathfrak{R}_4[\mathbf{u}] = \mathcal{U}$, see §4.2.8, suggests that we are indeed the result of an evolution of resolvers of the survival problem.

COMMENT Our species is Turing complete. Therefore we must explain the evolution of Turing completeness.

§6 Conclusion

§6.1 Purpose

¶1 · The problem theory is the union of set theory and computing theory. The integration of the two theories is achieved by using a new vocabulary to refer to old concepts, but mainly by giving the old theories a purpose that they did not have: to resolve problems. For example, a set defined by intension is named a problem, and the same set defined by extension is named its set of solutions. While both still refer to the same set, as it is the case in set theory, the status of each of them is now very different: one is a question and the other is an answer. And when the problem theory states that computing is resolving, it is calling a set resolvable if it is recursively enumerable, but mainly it is saying that the transition from intension to extension has to be calculated, because it is not written magically in “The Book”; someone has to write it.

¶2 · The purpose of resolving problems is not final, but the main conclusion of the paper, the Thesis §5.7.5, is nearly ultimate: We are Turing complete subjects because we are the result of an evolution of resolvers of the survival problem. In other words, we resolve problems to survive. So, if survival is indeed the ultimate purpose, then the problem theory provides purpose and meaning to set theory and to computing theory.

¶3 · The final Thesis §5.7.5 also closes a loop, because a Turing complete resolver $\mathfrak{R}_4[\mathbf{u}]$ can model *everything*, and then *everything* can be a solution, as it is stated in Theorem §3.2.1. But those *everything*s are not absolute, but limited to what is computable, see §4.4.11. That is, if Turing’s thesis stands, see §4.3.1, then *everything* is everything that is computable. This way a restriction of computing theory, countability, is inherited by problem theory and transferred to set theory; see the details below in Subsection §6.2. The other question that requires some more elaboration is the status of the Turing’s thesis itself, which we will postpone until Subsection §6.3.

¶4 · Nevertheless, besides that main Thesis §5.7.5, the problem theory concepts presented in this paper can be used to model, understand, and classify both natural and artificial resolvers, because the paper provides definitions, theorems, and taxonomies for resolvers, and also for problems. And, by the way, the paper defines adaptation, perception, and learning, and it shows that there are just three ways to resolve any problem: routine, trial, and analogy.

§6.2 Countability

¶1 · In computing everything is countable, see §4.1.14, and the problem theory in Turing universes inherited countability from computing theory, see §4.3.16. In a Turing universe, see §4.3.3, the limits of calculation are the limits of computing, and then there are only computable functions, including predicates, see §4.1.9, and computable sets, see §4.4.3. Then the problem theory in Turing universes is consistent if and only if computing is consistent. And computing is consistent, as a corollary to Church-Rosser theorem in λ -calculus, see Curry & Feys (1958) Chapter 4.

¶2 · Therefore, our way to control paradoxes in set theory, and then in this paper, is to confine ourselves to Turing universes. But don't worry; if this is a Turing universe, as it seems to be, then we are only excluding imaginary universes.

¶3 · For example, the mathematical theorem that states that everything is a solution is proved, and it makes sense, see §3.2.1. But it also causes paradoxes, because from it we derive $\mathbb{P} \subset \mathbb{S}$, but $\mathbb{P} \stackrel{1}{=} (\mathbb{S} \rightarrow \mathbb{B}) \stackrel{2}{=} 2^{\mathbb{S}}$, and then $|\mathbb{P}| \stackrel{3}{=} |2^{\mathbb{S}}| \stackrel{4}{=} 2^{|\mathbb{S}|} > |\mathbb{S}|$, by Cantor's theorem. It is not a paradox in a Turing universe because the forth equality is false in it. The second equality is false in a Turing universe because, as we saw in Lemma §4.4.8, there are resolvable problems that are not expressible, so $(\mathbb{S} \rightarrow \mathbb{B}) \subsetneq 2^{\mathbb{S}}$. The third equality is true, though it follows the second one! And the forth equality is false in a Turing universe because the number of computable sets is countable, so, if $|\mathbb{S}^*| = \aleph_0$, then $|2^{\mathbb{S}}|^* = \aleph_0 < 2^{\aleph_0} = 2^{|\mathbb{S}^*|}$, that is, $|2^{\mathbb{S}}| < 2^{|\mathbb{S}|}$. Therefore, $\mathbb{P}^* \subset \mathbb{S}^*$ is true, but \mathbb{P}^* is the set of computable predicates, that is, $\mathbb{P}^* = \mathcal{E}$ of §4.5.8, and $[2^{\mathbb{S}}]^* \subset \mathbb{S}^*$ is also true, but $[2^{\mathbb{S}}]^*$ is the set of computable sets, that is, $[2^{\mathbb{S}}]^* = \mathcal{R}$ of §4.5.8. The conclusion is that \mathbb{S}^* , the set of solutions, is the set of everything that is computable.

¶4 · We have just rejected the uncountable case, where $|\mathbb{P}| > \aleph_0$, but there are two other possibilities: the (infinite) countable case, where $|\mathbb{S}^*| = |\mathbb{P}^*| = |\mathbb{R}^*| = \aleph_0$, see §4.3.16; and the finite case, where $|\mathbb{S}_F| < |\mathbb{P}_F| < |\mathbb{R}_F| < \aleph_0$, see §5.1.7.

¶5 · We are finite, so it would be natural to restrict our investigations to the finite case, calling for finiteness instead of calling for countableness. But the finite case is trivial, and more importantly, the difference between an unrestricted universal computer and a finite universal computer is not qualitative but quantitative. There is not any step of any calculation that an unrestricted universal computer can compute and a finite universal computer cannot compute, see §4.2.10. So in the limit, that is, without time nor memory restrictions, we are universal computers. And note that those restrictions are variable, and that they can be relaxed nearly as desired just spending some more time, or building a faster computer machine, or using some more external memory. In the case of a Turing machine, the external memory is the tape, and the internal memory is where its processor keeps the internal state, see §4.1.3. Note also that we can code a program to generate every natural number, although we cannot follow the computation till its end. Summarizing: we are better defined saying that we are qualitatively universal Turing machines, but with some unspecified quantitative limitations, than saying that we are qualitatively finite state automata, because finite state automata are not expandable.

¶6 · Finally, the rejections of finiteness and uncountableness imply that countableness is the golden mean. This is Pythagorean heaven revisited, *everything is countable*, but this time we have rescued the terrifying $\sqrt{2}$, and other irrational numbers. As Kronecker said: "God made counting numbers; all else is the work of man".

§6.3 Intuition

¶1 · Is it possible to resolve a non-computable problem? A problem is computable if, by definition, see §4.4.3 and §4.4.4, a Turing machine can execute a valid resolution of the problem, so the non-computable problem would not be resolved by computing, but by other means. My answer to the question is ‘no’, because I think that a problem is resolvable if, and only if, the problem is computable, see §4.3.12.

¶2 · Nevertheless you may think otherwise, and say that there is another way of resolving, let us call it ‘intuition’, that is not computable. If that were the case, then the problem theory with its mathematical formulation, as presented in this paper, would capture the concept of ‘computable problem’, but not the whole concept of ‘problem’. In order to see this, please consider the following two statements:

- Some problems are computable.
- A universal computer can execute any computable resolution.

Even if you believe that there are resolvable problems that are not computable, you can still decide easily that both are true; the first is a fact, and the second is a theorem. And then everything in this paper would still be true of computable problems, computable resolutions, and computable solutions.

¶3 · The key point in this discussion is that ‘intuition’ would refute Turing’s thesis, see §4.3.1, because if there were ‘intuitive’ resolutions, then we could effectively calculate what is not computable. Turing’s thesis is not a theorem, and we follow Post (1936) in considering Turing’s thesis to be a law of nature that states a limitation of our own species calculating capacity, see Casares (2016a), by which we are bound to see ourselves as a final singularity. Summarizing: If Turing’s thesis were eventually false, then this problem theory would be about computable problems. But, while Turing’s thesis remains valid, the problem theory is about problems, the set of effectively calculable functions is countable (§4.3.1), universal computers are the most capable computing devices (§4.3.4), everything is an expression (§4.3.10), resolving is computing (§4.3.12), and the problem theory is complete (§5.7.3).

References

- Cantor (1895): Georg Cantor, “Contributions to the Founding of the Theory of Transfinite Numbers (First Article)”; in *Mathematische Annalen*, vol. xlvi, pp. 481–512, 1895. Translated by P.E.B. Jourdain in *Contributions to the Founding of the Theory of Transfinite Numbers*, Dover, New York, 1955. ISBN: 978-0-486-60045-1.
- Casares (2016a): Ramón Casares, “Proof of Church’s Thesis”; [arXiv:1209.5036](#).
- Casares (2016b): Ramón Casares, “Syntax Evolution: Problems and Recursion”; [arXiv:1508.03040](#).
- Curry & Feys (1958): Haskell B. Curry, and Robert Feys, with William Craig, *Combinatory Logic*, Vol. I. North-Holland Publishing Company, Amsterdam, 1958. ISBN: 978-0-7204-2207-8.
- Gandy (1980): Robin Gandy, “Church’s Thesis and Principles for Mechanisms”; DOI: [10.1016/s0049-237x\(08\)71257-6](#). In *The Kleene Symposium* (editors: J. Barwise, H.J. Keisler & K. Kunen), Volume 101 of Studies in Logic and the Foundations of Mathematics; North-Holland, Amsterdam, 1980, pp. 123–148; ISBN: 978-0-444-55719-3.
- Kleene (1936): Stephen C. Kleene, “ λ -Definability and Recursiveness”; in *Duke Mathematical Journal*, vol. 2, pp. 340–353, 1936, DOI: [10.1215/s0012-7094-36-00227-2](#).
- Lettvin et al. (1959): Jerome Y. Lettvin, Humberto R. Maturana, Warren S. McCulloch, and Walter H. Pitts, “What the Frog’s Eye Tells the Frog’s Brain”; in *Proceedings of the IRE*, vol. 47, no. 11, pp. 1940–1951, November 1959, DOI: [10.1109/jrproc.1959.287207](#).
- Post (1936): Emil L. Post, “Finite Combinatory Processes — Formulation 1”; in *The Journal of Symbolic Logic*, vol. 1, no. 3, pp. 103–105, September 1936, DOI: [10.2307/2269031](#). Received October 7, 1936.
- Post (1944): Emil L. Post, “Recursively Enumerable Sets of Positive Integers and their Decision Problems”; in *Bulletin of the American Mathematical Society*, vol. 50, no. 5, pp. 284–316, 1944, DOI: [10.1090/s0002-9904-1944-08111-1](#).
- Turing (1936): Alan M. Turing, “On Computable Numbers, with an Application to the Entscheidungsproblem”; in *Proceedings of the London Mathematical Society*, vol. s2-42, issue 1, pp. 230–265, 1937, DOI: [10.1112/plms/s2-42.1.230](#). Received 28 May, 1936. Read 12 November, 1936.
- Turing (1937): Alan M. Turing, “Computability and λ -Definability”; in *The Journal of Symbolic Logic*, vol. 2, no. 4, pp. 153–163, December 1937, DOI: [10.2307/2268280](#). Received September 11, 1937.
- Turing (1938): Alan M. Turing, “Systems of Logic Based on Ordinals”; Princeton University PhD dissertation. Submitted 17 May, 1938. Oral examination 31 May, 1938. Printed in *Proceedings of the London Mathematical Society*, vol. s2-45, issue 1, pp. 161–228, 1939, DOI: [10.1112/plms/s2-45.1.161](#). Received 31 May, 1938. Read 16 June, 1938.